2018 IEEE Computer Society Annual Symposium on VLSI

End-to-End Industrial Study of Retiming

Cunxi Yu[14], Chau-Chin Huang[2], Gi-Joon Nam[3], Mihir Choudhury[3], Victor N. Kravets[3],
Andrew Sullivan[3], Maciej Ciesielski[1], Giovanni De Micheli[4]
University of Massachusetts Amherst, MA, USA[1]
National Taiwan University, Taiwan[2]
IBM T.J. Watson Research Center, NY, USA[3]
EPFL, Switzerland[4]

Abstract -
Sequential circuits are combinational circuits that are separated by registers. *Retiming* is considered as the most promising technique for optimizing sequential circuits, that involves moving the edge-triggered registers across the combinational logic without changing the functionality. Despite significant efforts spent on sequential optimization since 1980's, there are few works discussed its performance in an end-to-end design flow. The retiming algorithms were mostly evaluated at the logic level. However, it turns out that the retiming results at logic level could be significantly different than evaluating the physical level.

This paper provides the findings of how retiming algorithms perform in an end-to-end industrial design flow, with seven industry designs taken from a recent 14nm microprocessor. Experiments are conducted with several complete industrial design flows. The evaluations are made at the end of the physical design flow. The experimental results show that the performance (design quality) of the retiming algorithms vary on the designs. Based these experimental results, we discover a feature that describes the retiming potentials of sequential designs. This model successfully forecast whether the given industrial designs could be significantly improved by retiming in an end-to-end design flow, regarding timing, area, and power.

Keywords—Sequential optimization, retiming, physical design, retiming prediction

I. Introduction

Retiming is a sequential optimization technique that has been studied since 1980's. Retiming techniques optimize the sequential circuits by relocating *edge-triggered registers*[1] across the combination logic without changing the design functionality. A lot of research efforts have been spent on developing promising retiming techniques that mainly target on three objectives:

- *min-delay*: minimize the worst-path delay of the circuits [1];
- *min-area*: minimize the number of registers of the circuits [2];
- *constrained min-area*: minimize the number of registers with a given worst-path delay constrain [3].

Numerous techniques have been proposed in our community to achieve these three objectives [1][2][4][5][6][7], and have been demonstrated with encouraging results. Although retiming assumes that the topology of the combinational logic is fixed, the quality (at logic-level) of the design can be further improved by combining the combinational logic optimization techniques and technology mapping [8][9][10]. In practice, constrained min-area retiming, has been incorporated into end-to-end design flows, which targets on improving the performance of the design regarding the delay, power, area, etc.

In 1997, N. Shenoy published the first retiming survey [11]. This work reviewed the theories and practical implementations of retiming, and the side issues of incorporating in the design flow. Due to the significant changes in the technology and design complexity, an up-to-date industrial study of retiming is necessary. Moreover, to our knowledge, no credible work ever evaluates the retiming algorithms in an end-to-end design flow. Most existing retiming algorithms were evaluated at the end of the logic synthesis, where the delay and area are measured after technology mapping or using unit delay-area models. However, due to the significant increase in design complexity and design rules, the gate-level netlist does not correlate well with the physical netlist [12][13]. Hence, in this work, the experimental results are collected at the end of the physical design process. The overview of the design flow is shown in Figure 1. Note that the negative retiming operations[2] are forbidden in our experiments. It turns out that the performance improvements gained by retiming evaluated the logic level could make the final physical netlist worse. Also, there are significant extra design efforts required for retimed designs, e.g., sequential equivalence checking (SEC). Thus, for the designs that retiming does not provide enough improvements in design performance, retiming needs to be avoided in the design flow. These give us the motivation for developing a *prediction* mechanism for retiming. The state-of-the-art retiming algorithms are reviewed in Section 2. The main difference compared to [11] is the min-area algorithm.

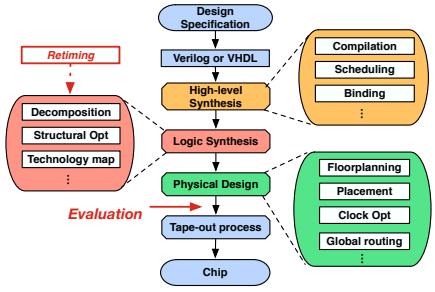

Fig. 1. Design flow used in this study.

Specifically, the main contributions of this paper are as follows:

- The complete end-to-end industrial study of retiming is shown in Section 3, using seven industrial designs are taken from the most recent microprocessor design with 14nm CMOS technology. Four different flows are tested, including the baseline flow and three retiming flows with different retiming options. The baseline flow is using the concept flow shown in Figure 1 without retiming applied. Retiming algorithms are embedded in the logic synthesis process, before technology mapping.
- The primary focus of the evaluations is the *design-quality*. Evaluations of retiming are made by comparing the results using a set of *timing, area* and *power* metrics, that are measured

[1]In the rest of this paper, *register* is used to represent *edge-triggered registers*.

[2]For min-delay retiming, negative retiming refers to ″the delay of critical path increases″; for min-area retiming, it refers to ″the number of registers increases.″

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00046

at the end of physical design with simulation. Based on the analysis of how retiming algorithms work, three classes of experiments have been identified: a) designs significantly improved by retiming; b) designs significantly disimproved by retiming, and c) retiming does not make many differences.

- The side challenges of retiming to the entire design flow are reviewed in Section 3-B. Specifically, the effects of retiming in physical design and sequential verification are discussed. These challenges with the unpredictable results shown in Section 3 motivate us in developing a prediction mechanism for retiming.
- In Section 4, a numerical model is introduced for prediction. This model describes a *sequential feature* of designs at the logic level that could include Boolean gates and high-level blocks such as Adders. We demonstrate that this model can be used for forecasting whether a design could be improved by retiming in an end-to-end design flow, with the seven industrial designs (Fig. 3).
- We evaluate the academic sequential benchmarks taken from *ISCAS-89* and *ITC99* using the proposed prediction model. It turns out that these benchmarks are not sufficient for evaluating retiming algorithms.

II. Background

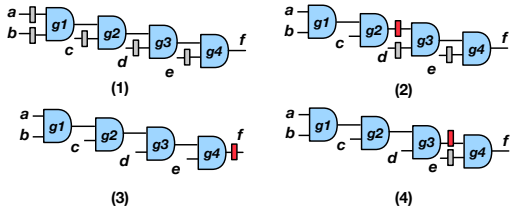

Fig. 2. Illustrative examples of retiming: 1) original netlist; 2) *min-delay* retiming; 3) *min-area* retiming; 4) *min-area* retiming under delay constrain (delay≤3).

TABLE I. Retiming results of examples in Figure 2.

	Baseline	*min-delay*	*min-area*	*min-area and d<3*
Delay (d)	4	2	4	3
Num. of Regs	5	3	1	2

The concept and the three objectives of retiming are illustrated using a simple example shown in Figure 2. The original design is shown in Figure 2-(1). We assume all the gates have unit delay one, and the edge-triggered registers are represented using the rectangles. The original design has five registers (n=5) and the delay of the critical path, $\{a,b\} \rightarrow f$, is four (d=4).

The *min-delay* retiming result is shown in Figure 2-(2). There are two iterations in this retiming: 1) move the two registers connected with a and b forward, which makes gate g_2 retimable; 2) move the retimed register and the register connected with c forward. The delay of the retimed design in Figure 2-(2) is two (d=2), and the number of registers is three (n=3). The *min-area* retiming result is shown in Figure 2-(3). This solution requires two more iterations in addition to the solution of min-delay retiming, which move the registers all the way to primary output and reduce the number of registers to one. The delay is increased to four. The third objective is min-area retiming under delay constrain. In this example, let the delay constraint be $d \leq 3$. The solution is shown in Figure 2-(4). The comparison of these three retiming solutions to the original design is shown in Table I. We can see that the min-delay retiming gives the best delay solution ($d = 2$), min-area offers the minimum number of registers, and min-area retiming with delay constraints gives balanced results in between of min-area and min-delay.

A. Formulation of Retiming

Most logic optimization techniques are formulated based on direct graph representation. The logic netlist, so called *Boolean network* can be modeled using direct graph $G(V, E)$, where each vertex v corresponds to a logic gate g in the design. Besides constructing the direct graph directly from the netlist, this can also be done based on the transformed Boolean network, such as *And-Inv-Graph* [14] and its sequential version [15]. In the context of retiming, the sequential Boolean network is the combinational network separated by the memory elements, which are assumed to be ideal registers. The edges in the graph $G(V, E)$ represent the interconnections of the logic gates in the design.

Let us denote e_{uv} is an edge of $G(V, E)$, e_{uv}: $u \rightarrow v$, and w_{uv} is the weight of the edge e_{uv} which represents the number of registers between the two vertex u and v. The weight of the edges directed from and into the primary inputs (PIs) and primary outputs (POs) is zero. Each vertex v in $G(V, E)$ represents its delay of the corresponding gate, denoted $d(v)$. The problem of retiming is denoted by retiming lag function $r(v)$ [16]: $V \rightarrow Z$. Let us denote that w^r_{uv} is the weight of the edge e_{uv}. For any retiming, it can be represented by Eq. 1.

$$w^r_{uv} = r(v) - r(u) + w_{uv} \tag{1}$$

The value of $r(v)$ represents the movement of the registers for vertex v. If it is forward retiming (from inputs to outputs), then $r(v)$ is a negative. For any legal retiming, the condition shown in Eq. 2 must be satisfiable.

$$w_{uv} + r(v) - r(u) \geq 0 \tag{2}$$

B. Min-delay Retiming

The min-delay retiming problem is as follows: Given $G(V, E)$ with a vertex delay function d and edge weight function w, find a legal retiming r, such that the cycle time c is minimized:

$$c = \max_{p:w_r(p)=0} \{d(p)\}, \tag{3}$$

where $d(p)$ is the path delay, and $w_r(p)$ is the retimed register count on the path p. To this problem, Leiserson and Saxe [16] developed a classic algorithm: Two matrices W and D are first defined as:

$$W(u, v) = \min_{p:u \rightsquigarrow v} \{w(p)\}, \tag{4}$$

$$D(u, v) = \max_{p:u \rightsquigarrow v \wedge w(p)=W(u,v)} \{d(p)\}, \tag{5}$$

$W(u, v)$ gives the minimum register count on any path from u to v. $D(u, v)$ determines the maximum delay from u to v for the minimum register count. The two matrices can be obtained by solving an all-pairs shortest paths problem in G. Afterward, a binary search for the minimum clock cycle is performed. In each iteration, a Bellman-Ford algorithm can be employed to test whether a legal retiming exists with the current cycle time c. The algorithm above runs in $O(V^3 \lg V)$ time because each iteration costs $O(V^3)$ time for a Bellman-Ford algorithm and the binary search runs in $O(\lg V)$.

Leiserson and Saxe [16] also proposed another more efficient relaxation algorithm, which runs in $O(VE)$ for examining if a retiming exist for a given clock cycle c. A function $\Delta(v)$ gives the largest delay seen from any path that terminates at the output of v:

$$\Delta(v) = d(v) + \max_{u \in FI(v),\ w(e_{uv})=0} \{\Delta(u)\}. \tag{6}$$

Therefore, the cycle time can be expressed as follows:

$$c = \max_{v \in V} \{\Delta(v)\}. \tag{7}$$

The relaxation algorithm consist of alternately updating the functions $\Delta(v)$ and $r(v)$ for $|V|-1$ times. The optimality is guaranteed because each iteration simulates a pass off a Bellman-Ford algorithm (i.e., a vertex being relaxed in a pass of the Bellman-Ford algorithm must be updated in an iteration of the relaxation algorithm), one can obtain a feasible retiming under the target cycle time c, if it exists. Because calculating $\Delta(v)$ in an iteration costs $O(E)$ time, the relaxation algorithm runs in $O(VE)$ time. Later on, the runtime of this algorithm was improved by Shenoy and Rudell [17] by adding an early break mechanism.

C. Min-area Retiming

The typical min-area retiming refers to minimizing the number of registers without delay constraints. In which case, the problem can be formulated as a minimum-cost flow problem using linear programming. The formulation is as follows:

$$\min : \sum_{\forall e_{uv}} r(u) - r(v) \wedge (\forall e_{uv}, r(u) - r(v) \leq w_{uv}) \tag{8}$$

Goldberg [5] presented a practical push-relabel method that can solve this problem in $O(V^2\text{E log(VC)})$ worst-case runtime, where V is the number of vertices, E is the number of edges, and C is the maximum cost of the edges. A. Hurst et al. [2] proposed a the min-area retiming using maximum network flow problem. It turns out that within a combinational network, minimizing the number of registers by retiming is equivalent to finding a minimum cut. Note that the minimum cut problem is the dual of the maximum network flow problem. Computing maximum flow of a network is much less complex than minimum cost determination. Although there may exist many minimum cuts, that approach always generates one minimum cut that provides the minimum number of movements of the registers. This is claimed to simplify the computation of the initial states and minimize the side effects of the design [2]. The worst-case runtime of maximum flow approach is bounded by $O(R^2\text{E})$, where R is the initial number of registers, and E is the number of edges. This algorithm requires repeated iterations while the number of iterations is typically small, as demonstrated by the authors.

D. Constrained Min-area Retiming

Although it is claimed that pruning the redundant storage elements in the design reduces the area, power, and verification cost, most designs request a specific target clock period. The first attempt of constrained min-area retiming was presented by Shenoy et al. [17]. The implementation was composed by computing the W and D matrices, and the minimum cost circulation implementation. Let us denote $N_{fanin}(v)$=number of $fanins$ of vertex v, $N_{fanout}(v)$=number of $fanouts$ of vertex v. The formal definition of this problem is represented as follows [11]:

$$\min : \sum_{\forall v} |N_{fanin}(v) - N_{fanout}(v)| \cdot r(v) \tag{9}$$

$$\wedge\ \forall e_{uv}, r(u) - r(v) \leq w_{uv} \tag{10}$$

$$\wedge\ \forall e_{uv}, r(u) - r(v) \leq W(u,v) - 1 \tag{11}$$

Equation 9 represents the cost of number of registers of all register relocations. Equation 10 and 11 constrain each relocation must be legal retiming and under the delay constrain. We can see that this problem can be solved by combining the algorithms proposed in Section 2.2 and Section 2.3. The most recent constrained min-area retiming method was proposed by A. Hurst et. al [3], which the min-area appoach is based on the work of [2]. That approach is developed based on the observation that area-critical and timing-critical regions are rare overlapped. In that work, the timing constrains of retiming requires the exiting min-delay retiming algorithms, which give the initial register positions for their min-area appoach.

III. Experimental Results

The results used for this study are shown in Table II. They are obtained with seven industry designs taken from the recent 14nm microprocessor design, with total twenty-eight *end-to-end* experiences. Each design is applied with four different complete design flows. *baseline* uses the flow shown in Figure 1, which is considered as the benchmark flow (without retiming). The three retiming design flows are added with three different retiming options to the *basline* flow. *r1,r2,r3* are the three different combinations of retiming that provide the most effective results: 1) *r1*={min-area, min-delay, min-area}; 2) *r2*={min-area, min-delay}; and 3) *r3*={min-delay, min-area}. The retiming process is applied on NAND2-based network within logic synthesis process, while each gate is considered as a unit delay gate. The min-delay retiming is implemented by combining the relaxation algorithm [1], and the heuristic min-delay algorithm implemented in ABC [18]. The min-area retiming is implemented using constrained min-area retiming algorithm based on *max-flow min-cut* proposed in [3].

A. Evaluations of Retiming

Table II lists the selected experiences results. It includes the major metrics used for evaluating the design at physical level using the flow shown in Figure 1. The evaluations mainly target on *timing*, *area* and *power*. The timing metrics include: 1) *Worst Negative Slack* (WNS), 2) *Total Negative Slack* (TNS), 3) *Latch-to-Latch WNS* (l2lWNS), 4) *Latch-to-Latch TNS* (l2lTNS), and 5) number of negative paths (#NEG). The area metrics include the *number of registers* (#reg) and *physical area* (Area). The power metrics inlude *dynamic power* (Dpower) and *static power* (Spower). The CPU runtime of the complete design process is included in the last column. Note that the extra CPU time comparing to the *baseline* mostly come from physical design process. The CPU time of retiming in Table II are all less than 600 seconds.

Based on the results shown in Table II, the designs can be classified in three types:

- Type 1) retiming significantly improves the timing and/or area at the end of design flow, including *ibm6* and *ibm7*;
- Type 2) these three retiming options all make the timing and/or area worse compared to baseline, including *ibm1* and *ibm2*;
- Type 3) retiming may make the timing and/or area worse than baseline, including *ibm3*, *ibm4* and *ibm5*.

In Table II, all the retiming operations are **positive** retiming, i.e., for min-delay, the critical path delay at the logic level always decreases or remain the same as before retiming. For min-area, the number of registers always decreases or remain the same, and the critical path delay is always less or equal to the given delay constraint. Any negative retiming operations will terminate the retiming process and return the original netlist.

Specifically, for ibm6 and ibm7, all the three retiming options give improvements in timing (mostly TNS) and area comparing to baseline. One observation is that while adding more registers in the design, the area and power may not increase. This is because adding more registers improve the timing, which timing closure requires fewer efforts during physical design process. For designs ibm3, ibm4 and ibm5, retiming doesn't give promising improvements compared to

TABLE II. EVALUATION OF THREE RETIMING OPTIONS IN THE END-TO-END DESIGN FLOW. BOLD RESULTS INDICATE THE BEST RESULTS AMONG THE FOUR FLOWS.

Design	Options	WNS	TNS	l2lWNS	l2lTNS	#NEG	#reg	Area	Dpower	Spower	CPU Time(hr)
ibm1	baseline	**-14.491**	**-10497**	**-17**	**8.607**	**1326**	3214	340230	26.9089	**18.8614**	20.3
	r1	-23.425	-23170	-44	7.007	2156	**3095**	336509	25.8387	20.4603	20.9
	r2	-14.75	-15169	-47	7.260	1668	3193	337795	26.8823	19.9236	21.0
	r3	-16.68	-18651	-21	8.151	1775	3113	**335646**	**25.2462**	19.9465	21.0
ibm2	baseline	**-23.87**	**-3275**	**-50**	8.151	**641**	8115	**836771**	**111.0640**	**16.5280**	13.6
	r1	-36.524	-3894	-101	7.548	775	**7901**	897393	115.8500	17.7728	22.8
	r2	-37.397	-4340	-83	**8.200**	778	8082	893571	115.5700	17.5046	22.8
	r3	-35.869	-4118	-83	8.099	772	8247	899329	118.5860	17.5122	22.8
ibm3	baseline	-17.884	-4882	-8	8.504	670	2672	216950	45.5289	9.7992	6.9
	r1	-17.142	-4899	-6	8.705	664	**2485**	**209388**	**43.1193**	10.0417	21.1
	r2	**-14.711**	**-4662**	-2	**9.569**	**646**	2633	213527	44.7554	**9.5785**	21.2
	r3	-19.675	-4976	-4	8.947	685	2567	212362	44.1308	9.9634	21.1
ibm4	baseline	-20.507	-13074	-9064	-16.317	1110	4186	509862	119.9100	28.7540	20.8
	r1	-23.922	-17110	-13960	-21.856	1138	4186	512716	120.9800	31.1402	21.9
	r2	-22.295	-14784	-10626	-21.196	1132	**3954**	517506	120.4960	30.5482	21.9
	r3	**-18.965**	**-9919**	**-6288**	**-12.797**	**892**	4010	**506887**	**118.4360**	**28.4195**	21.9
ibm5	baseline	-20.675	-2584	-55	2.437	471	4682	514430	109.7720	11.8720	19.7
	r1	-49.791	-3229	-62	2.234	1163	**4597**	**503435**	**108.8470**	**11.1319**	20.5
	r2	**-19.928**	**-1735**	-46	4.978	**203**	4684	505298	112.2140	11.3123	20.6
	r3	-54.379	-3266	**-22**	**9.143**	1091	4622	507626	109.6960	11.3450	20.6
ibm6	baseline	-366.635	-14327	0	9.992	365	8238	812092	39.3554	5.378	19.3
	r1	-367.179	-9412	0	10.000	303	**7423**	**711125**	**35.213**	**4.779**	21.6
	r2	**-363.009**	-9860	0	10.000	314	8241	744181	39.4629	4.9856	21.7
	r3	-363.632	**-6982**	0	10.000	**140**	8657	769391	41.3688	5.089	21.8
ibm7	baseline	-55.537	-38388	-18005	-45.000	1732	2846	446838	134.214	24.7752	1.3
	r1	-56.131	-35473	-15361	-42.810	1736	2819	438761	131.735	23.9898	1.5
	r2	-53.184	-33876	-13653	-39.838	1740	**2625**	**428183**	127.636	24.1353	1.4
	r3	**-45.805**	**-31904**	**-12098**	**-38.850**	**1620**	2688	429153	**127.585**	**23.6245**	1.9

baseline. And, the performance of retiming varies on the combination and the order of retiming algorithms applied. For example, ibm5, r2 gives almost the same result compared to baseline, but r1 and r3 make the timing much worse. For the designs ibm1 and ibm2, all the three retiming options make the timing and area worse than baseline. We didn't consider the runtime of retiming in those comparisons since retiming takes little time compared to other design automation processes. In summary, the results in Table II show that:

- 1) The performance of retiming varies on different designs.

For example, ibm6 and ibm7 are improved by all retiming options, but ibm1 and ibm2 are worse than baseline with any retiming option. The main reason is that delay-driven retiming algorithms target on minimizing the critical path delay, i.e., the levels of the longest combinational paths. However, the logic-level critical paths could be very different to the critical paths at the physical level. This means that, even though there are significant reductions provided by retiming at the logic level, the critical path delay at physical level could be worse. This directly affects the timing closure, such as WNS and TNS. For timing closure, more physical design efforts are required to meet the requirements of timing constraints. On the other hand, the area and CPU time could also increase, e.g., the routability decreases after retiming. This provides the main motivation for pre-analysis of retiming to predict if a given design could be improved by retiming.

- 2) The combination and ordering of retiming approaches affect the timing, area, and power, even though the statistics (area and levels) at logic level are the same.

For most industry designs, retiming is required to provide balanced performance regarding timing, area, power, etc., which is the retiming objective reviewed in Section II-D. The constrained min-area retiming methodology hasn't been changed in the last twenty years, which was proposed by Shenoy et al. [17], using a combination of multiple min-delay and min-area retiming. This method works for many designs based on the observation that the regions of area-critical and timing-critical rarely overlap [3]. However, although the optimum solutions can be found, the retiming operations could be very different. For example, the uniqueness of the maximum flow does not imply the uniqueness of the minimum cut. One observation in this study is that the number of retiming depths of retimed registers could be very different using different retiming options, even though the statistics at the logic level are the same. Hence, the varieties of retiming solutions, and the mismatch between logic and physical criticalities make the choice of retiming options crucial.

- 3) For the designs that can be improved by one of the retiming options, they are likely to be verified by all the retiming options.

One observation is that the designs that cannot be verified by any one retiming option, they are unlikely improved by other retiming options. Some designs are significantly disimproved. On the other side, the designs are improved by one retiming option; they are likely to be improved by other retiming options as well. For example, designs *ibm6* and *ibm7* are improved significantly by all the retiming options. This gives us the main motivation of developing a retiming prediction model to pre-analyze whether retiming algorithms are effective for a given design, which is shown in Section IV.

B. Challenges of Retiming

The challenges of retiming are considered in three parts: a) retiming complexity, b) retiming performance in design-quality; and c) the side effects. The complexity of retiming has been discussed in [11] and briefly reviewed in Section 2. Hence, in this section, we focus on analyzing the last two. As shown in Table II, the design performance of the physical netlist is not guaranteed to be improved by retiming algorithms. Since all the retiming operations undertake to reduce the critical path delay and area of the design at the logic level, these results also show that there are no strong correlations between logic and physical netlist. This is believed to be worse as the number of design rules increasing for the advanced technology nodes. Apparently, this has significant impacts on the design-quality and Time-to-Market. To forecast a given design whether its performance could be improved by retiming becomes extremely important.

One of the main side effects of the design flow is sequential verification. It is stated that the verification of retimed circuits could be solved very efficiently and takes $O(|E|)$ time [11], where verification

TABLE III. DEMONSTRATION OF THE PROPOSED PRE-ANALYSIS MECHANISM USING ONE DESIGN IN EACH EXPERIMENTAL CATEGORY PRESENTED IN SECTION 3.1. THE RESULTS ARE SHOWN IN PERCENTAGE COMPARED TO THE RESULTS OF $baseline$.

Design	B_{foward}	$B_{backward}$	Options	WNS	TNS	l2lWNS	l2lWNS	Area	Power
ibm1	0.118	0.044	r1	1.617	2.207	2.588	0.814	0.989	1.085
			r2	1.018	1.445	2.765	0.843	0.993	1.056
			r3	1.151	1.777	1.235	0.947	0.987	1.058
ibm2	0.041	0.022	r1	1.530	1.189	2.020	1.080	1.072	1.047
			r2	1.566	1.325	1.660	0.994	1.067	1.042
			r3	1.502	1.257	1.660	1.01	1.074	1.066
ibm3	0.33	0.041	r1	0.959	1.003	-	1.024	0.965	1.025
			r2	0.823	0.955	-	1.125	0.984	0.977
			r3	1.100	1.019	-	1.052	0.979	1.017
ibm4	**0.616**	0.035	r1	1.166	1.308	1.540	**0.746**	1.005	1.023
			r2	1.087	1.130	1.172	**0.769**	1.014	1.016
			r3	**0.925**	**0.758**	**0.694**	1.275	**0.994**	**0.987**
ibm5	0.045	0.001	r1	2.408	1.250	1.127	0.917	0.979	0.938
			r2	**0.964**	**0.671**	0.836	2.043	0.982	0.953
			r3	2.630	1.264	0.400	3.752	0.987	0.956
ibm6	**5.512**	0.082	r1	1.001	**0.659**	-	1.000	**0.875**	**0.894**
			r2	**0.990**	**0.688**	-	1.000	**0.916**	**0.993**
			r3	**0.992**	**0.487**	-	1.000	**0.947**	1.038
ibm7	**0.763**	0.082	r1	1.011	**0.924**	**0.853**	**0.951**	**0.982**	**0.980**
			r2	**0.957**	**0.882**	**0.758**	**0.885**	**0.958**	**0.955**
			r3	**0.827**	**0.831**	**0.672**	**0.863**	**0.960**	**0.951**

is restricted to verify the design before and after retiming. However, the complete sequential verification is still needed, especially the typical retiming flow includes multiple iterations and usually combined with logic synthesis [19]. Despite the progress of the sequential verification, directly check the sequential equivalence is still extremely hard. For example, using model checking technique *IC3* [20], it can't directly verify the properties of retimed $s13207$, $s15850$, and $s38584$ ISCAS-89 designs, with more than 24 hours[3]. Engineering Change Order (ECO) is another side challenge of retiming. ECO for retimed circuits becomes *Sequential Engineering Change Order* (SECO), which has never been discussed. Although several techniques of combinational ECO proposed using formal methods and structural methods, determination of the minimal patches in combinational logic is still a challenging problem. Functional methods can find smaller patches but limited to its scalability. Structural methods improve the runtime but limited by the structural similarity.

IV. RETIMING PREDICTION

Due to the uncertainty of retiming and significant extra design efforts, a pre-analysis mechanism for predicting whether a design is potentially improvable by retiming is needed. The purpose of the proposed prediction mechanism is that *given a sequential circuit including Boolean gates, high-level blocks, and registers, it outputs a normalized value that represents the retiming potentials in an end-to-end design flow*. Based on a large number of experiences and analysis using industry and academic benchmarks, we observe that the retiming flow gives significant improvements when the pipeline stages are not well-balanced, regardless of the options of the retiming flow. However, it is not sufficient by just considering the pipeline stages. The reason is that *unbalanced registers*[4] may not be retimable. Hence, we introduce the *Balancing* metric. It is defined using Equation 12, where r_i is forward, or backward retimable register[5], N is the number of retimable registers in the design. Hence, for each design, it has two balancing value, $B_{forward}$ and $B_{backward}$. This metric is sensitive to the timing model since it is calculated at logic-level. For the blocks have multiple inputs and outputs (e.g., 4-bit adder), we consider that the pin-to-pin delay of all the paths is the same. For example, assuming delay of AND2 is one, in Figure 2 (a), $B_{backward}$=0 because there is no backward retimable registers; $B_{forward}$=$(4+4)/2 = 4$ since registers a and b are forward retimable.

$$B = \sum_{i=0}^{i=N-1} |Slack_{out}(r_i) - Slack_{in}(r_i)| \ / \ N \tag{12}$$

A. Performance Analysis via Balance Value

Table III lists the analysis of retiming performance using the proposed pre-analysis mechanism, with the designs shown in Table II. The forward and backward balancing values are listed in the first two columns. The timing, area, and power results are divided by the results generated from *baseline* experiences. In Table III, for any value > 1, represents negative result; for any value < 1, represents positive result. As the $B_{forward}$ (see Eq. 12) value increasing, the design is more likely to be improved by retiming. For example, for design *ibm2*, both $B_{forward}$ timing and $B_{backward}$ are smaller than 0.05, which means that the design around the retimable registers is well-balanced. Not spuriously, the area and power results generated from the flow using retiming, are all negative. For design *ibm4*, $B_{forward}$=0.616, which indicates that there are potential improvements could be done by retiming. We can see that positive retiming results appear for this design and $r3$ improves timing, area, and power simultaneously. For design *ibm6*, the $B_{forward}$=5.512, where we see that the design has been significantly improved by retiming. Note that the balancing value may be different using different timing model. However, the standardization remains regardless of the select timing model. We observe that the backward balance value are not sufficient to provide any evidence of analyzing the retiming performance.

To further analyze the forward balance value, the retiming, area and power related metrics reported by the design flow are all collected in Figure 3. The x-axis shows the forward balance value, and the y-axis shows the overhead/improvement provided by retiming. For example, the data point with y=1.5 means a 50% overhead for some metric. The results are classified into six groups: timing metrics improved/disimproved, area metrics improved/disimproved, and power metrics improved/disimproved. The retiming performance is considered as negative if some metrics have been significantly disimproved, or most of the metrics remaining the same. We can see that as the forward balance value increasing, the retiming becomes more effective. For the designs with small forward balance value, the area, timing and power metrics are $2\times$ worse.

[3]These experiments are obtained without the industrial design flow. They are tested using *pdr* command in ABC[18].

[4]Unbalanced registers refer to the registers with significant differences between the input and output Slack time.

[5]Definitions of forward and backward retiming refer to Equation 1 and 2.

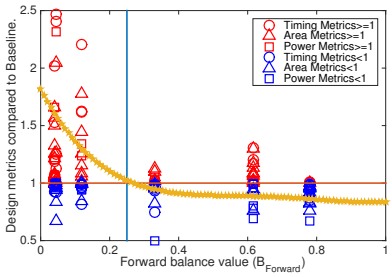

Fig. 3. Forward balance analysis with the designs listed in Table III.

In addition, we find that the most sensitive parameter in respect to forward balancing value is TNS. This means that the balancing value could be more effective in predicting the improvements of TNS. To demonstrate this, we create 120 benchmarks using high-level synthesis tool by manually modifying the location of the registers, such that the range of the balancing values of those benchmark is wide. We then obtain the TNS results of all the benchmarks using the same physical design flow. The results are shown in Figure 4. The x-axis represents the forward balancing value, and y-axis represents the percentage of TNS improvements. The data points above the horizontal line y=0 represent the designs get positive TNS improvements by retiming. We can see that the TNS improvements are clearly increasing as the forward balancing value increases.

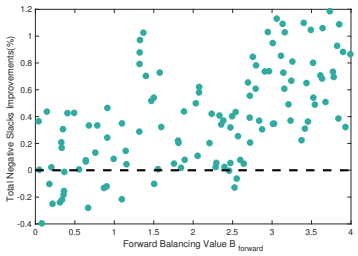

Fig. 4. Total negative slacks (TNS) improvements (%) versus forward balancing value of 120 designs.

Based on the results shown in Table III and Figure 4, we can see that the unbalanced designs are likely to be optimized by retiming. Due to the high extra design automation efforts, it is essential to understand if the designs are suitable for retiming in advance. Meanwhile, we can see that selecting different benchmarks could dramatically affect the evaluations of retiming algorithms.

V. Conclusion

This paper provides the first end-to-end industrial study of retiming, using the recent 14nm industrial designs. The state-of-the-art min-delay, min-area and constrained min-area retiming algorithms are evaluated in the complete design flows, with three retiming options. The results show that evaluations of retiming algorithms at logic level could be misleading. Due to the challenges of retiming discussed in this paper, a retiming-prediction model is introduced to forecast whether retiming algorithms could improve a given design. We demonstrate that this prediction model correctly predict the retiming potentials of the seven 14nm industrial designs. With this model, we are able to reduce the cost of designing an efficient flow in order to reduce *Time-to-Market*.

ACKNOWLEDGMENTS: The results are collected when Cunxi and Chau-Chin were interns at IBM T.J. Watson Research Center. This project is partially funded by ERC-2014-AdG 669354 grant.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp. 5–35, 1991.

[2] A. P. Hurst, A. Mishchenko, and R. K. Brayton, "Fast minimum-register retiming via binary maximum-flow," in *Formal Methods in Computer Aided Design, 2007. FMCAD'07.* IEEE, 2007, pp. 181–187.

[3] A. Hurst, A. Mishchenko, and R. Brayton, "Scalable min-register retiming under timing and initializability constraints," in *Proceedings of the 45th annual Design Automation Conference.* ACM, 2008, pp. 534–539.

[4] P. Pan, "Continuous retiming: Algorithms and applications," in *Computer Design: VLSI in Computers and Processors, 1997. ICCD'97. Proceedings., 1997 IEEE International Conference on.* IEEE, 1997, pp. 116–121.

[5] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of algorithms*, vol. 22, no. 1, pp. 1–29, 1997.

[6] D. P. Singh, V. Manohararajah, and S. D. Brown, "Incremental retiming for FPGA physical synthesis," in *Proceedings of the 42nd annual Design Automation Conference.* ACM, 2005, pp. 433–438.

[7] S. S. Sapatnekar and R. B. Deokar, "Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, pp. 1237–1248, 1996.

[8] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 74–84, 1991.

[9] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 63–73, 1991.

[10] J. Cong and C. Wu, "Optimal FPGA mapping and retiming with efficient initial state computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 11, pp. 1595–1607, 1999.

[11] N. Shenoy, "Retiming: Theory and practice," *Integration, the VLSI journal*, vol. 22, no. 1, pp. 1–21, 1997.

[12] F. Liu, Ed., *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016.* ACM, 2016.

[13] S. Parameswaran, Ed., *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017.* IEEE, 2017.

[14] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG Rewriting A Fresh Look at Combinational Logic Synthesis," in *43rd DAC.* ACM, 2006, pp. 532–535.

[15] A. Mishchenko and R. Brayton, "Recording synthesis history for sequential verification," in *Formal Methods in Computer-Aided Design, 2008. FMCAD'08.* IEEE, 2008, pp. 1–8.

[16] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming (preliminary version)," in *Third Caltech conference on very large scale integration.* Springer, 1983, pp. 87–116.

[17] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design.* IEEE Computer Society Press, 1994, pp. 226–233.

[18] A. Mishchenko *et al.*, "ABC: A System for Sequential Synthesis and Verification (2007)," *URL http://www. eecs. berkeley. edu/alanmi/abc*, 2010.

[19] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable sequential equivalence checking across arbitrary design transformations," in *Computer Design, 2006. ICCD 2006. International Conference on.* IEEE, 2007, pp. 259–266.

[20] F. Somenzi and A. R. Bradley, "IC3: where monolithic and incremental meet," in *FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, 2011, pp. 3–8.