VirtualSync+: Timing Optimization with Virtual Synchronization

Grace Li Zhang, Bing Li, Xing Huang, Xunzhao Yin, Cheng Zhuo, *Senior Member, IEEE*, Masanori Hashimoto, *Senior Member, IEEE*, Ulf Schlichtmann, *Senior Member, IEEE*

***Abstract*—In digital circuit designs, sequential components such as flip-flops are used to synchronize signal propagations. Logic computations are aligned at and thus isolated by flip-flop stages. Although this fully synchronous style can reduce design efforts significantly, it may affect circuit performance negatively, because sequential components can only introduce delays into signal propagations but never accelerate them. In this paper, we propose a new timing model, VirtualSync+, in which signals, specially those along critical paths, are allowed to propagate through several sequential stages without flip-flops. Timing constraints are still satisfied at the boundary of the optimized circuit to maintain a consistent interface with existing designs. By removing clock-to-q delays and setup time requirements of flip-flops on critical paths, the performance of a circuit can be pushed even beyond the limit of traditional sequential designs. In addition, we further enhance the optimization with VirtualSync+ by fine-tuning with commercial design tools, e.g., Design Compiler from Synopsys, to achieve more accurate result. Experimental results demonstrate that circuit performance can be improved by up to 4% (average 1.5%) compared with that after extreme retiming and sizing, while the increase of area is still negligible. This timing performance is enhanced beyond the limit of traditional sequential designs. It also demonstrates that compared with those after retiming and sizing, the circuits with VirtualSync+ can achieve better timing performance under the same area cost or smaller area cost under the same clock period, respectively.**

I. Introduction

IN DIGITAL circuit designs, clock frequency determines the timing performance of circuits. In the traditional timing paradigm, sequential components, e.g., edge-triggered flip-flops, synchronize signal propagations between pairs of flip-flops. Consequently, these propagations are blocked at flip-flops until a clock edge arrives. At an active clock edge, the data at the inputs of flip-flops are transferred to their outputs to drive the logic of the next stage. Therefore, combinational logic blocks are isolated by flip-flop stages. This fully synchronous style can reduce design efforts significantly, since only timing constraints local to pairs of flip-flops need to be met.

Within the traditional timing paradigm, timing analysis and timing optimization have been explored extensively. In timing analysis, researchers focus on improving the execution efficiency and accuracy of analysis [2]–[7]. In timing optimization, several methods have been proposed to improve timing performance. A widely adopted method is sizing, in which logic gates are sized to improve objectives such as clock frequency and area efficiency, while timing constraints between flip-flops are satisfied. Typical methods for gate sizing are based on Lagrangian Relaxation and sensitivity [8]–[11]. The second method to improve circuit performance in the traditional paradigm is retiming, which moves sequential components, e.g., flip-flops, but still preserves the correct functional behavior of circuits. The existing retiming methods usually focus on reducing execution time while improving the performance of digital circuits [12]–[16]. Useful clock skew is the third method to enhance timing performance of digital circuits. For example, [17]–[19] intentionally exploits clock skews to improve the yield of digital circuits. Beyond these techniques, a further method to improve clock frequency is to introduce approximation in computational result [20].

Wave-pipelining is the third method to improve circuit performance, where several logic waves are allowed to propagate through combinational paths without intermediate sequential components. Wave-pipelining paths are different from multi-cycle paths where there is only one logic wave propagating along them at a moment. In timing constraints, multi-cycle paths only restrict their delays to be smaller than a specified upper bound [21], [22] while wave-pipelining paths restrict not only the upper bound but also the lower bound of their delays. Wave-pipelining provides a mechanism to make the clock frequency of a circuit independent of the largest path delay, which limits circuit performance in traditional circuit designs [23]. As early as in [24], an algorithm to automatically equalize delays in combinational logic circuits is proposed to realize wave-pipelining. In [25], a linear method to minimize the clock period using wave-pipelining is proposed. This method is also explored for majority-based beyond-CMOS technologies to improve the throughput of majority inverter graph designs in [26]. Testing methods of wave-pipelined circuits are proposed in [27]. Recently, wave-pipelining is applied to accelerate the dot-product operations of neural network accelerators in [28]. In addition, wave-pipelining has been applied to enhance netlist security [29], [30].

The first two methods above can be used separately or jointly to improve circuit performance. However, sequential

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 146371743 – TRR 89 Invasive Computing.

A preliminary version of this paper was published in the Proceedings of Design Automation Conference (DAC), 2018 [1]. *(Corresponding author: Xing Huang)*

Grace Li Zhang, Bing Li, Xing Huang and Ulf Schlichtmann are at the Chair of Electronic Design Automation, Technical University of Munich (TUM), Munich 80333, Germany (e-mail: grace-li.zhang@tum.de; b.li@tum.de; xing.huang@tum.de; ulf.schlichtmann@tum.de).

Xunzhao Yin and Cheng Zhuo are with the College of Information Science and Electronic Engineering, Zhejiang University (e-mail: xzyin1@zju.edu.cn; czhuo@zju.edu.cn).

Masanori Hashimoto is with the Department of Information Systems Engineering, Osaka University, (e-mail: hasimoto@ist.osaka-u.ac.jp).

components are assumed to synchronize signal propagations in these methods, where no signal propagation through sequential components is allowed except at the clock edges. This synchronization with sequential components achieves many benefits such as reducing design efforts. However, it limits circuit performance in two regards. Firstly, sequential components have inherent clock-to-q delays and impose setup time. The former becomes a part of combinational paths driven by the corresponding flip-flops and the latter requires a further part of the timing budget for the critical paths. Secondly, delay imbalances between flip-flop stages cannot be exploited since signal propagations are blocked at flip-flops instead of being allowed to propagate through flip-flops. Although clock skew scheduling can relieve this problem to some degree, it still suffers the inherent clock-to-q delays and setup time constraints of flip-flops. The third method above, wave-pipelining, allows signals to pass through sequential stages without flip-flops. However, this technique is not compatible with the traditional timing paradigm. Although a wave-pipelining utility that interacts with commercial tools is proposed in [28] to achieve delay balance, the proposed method in this work can only deal with generic combinational circuits without feedback loops, which restricts its application in sequential digital circuits.

In this paper, we propose a new timing model, VirtualSync+, which removes the confines of the traditional timing paradigm. Our contributions are as follows:

- In the proposed new timing model, sequential components and combinational logic gates are both considered as delay units. Combinational logic gates add linear delays of the same amount to short and long paths, where sequential components provide non-linear delay effects, which provide different delay effects to fast and slow signal propagations.
- With the new timing model, a timing optimization framework is proposed to allocate sequential components only at necessary locations in the circuit to synchronize signal propagations, while the functionality of circuits is maintained. The absence of flip-flops at some sequential stages allows a virtual synchronization to provide identical functionality as in the original circuit. Consequently, the original clock-to-q delays and setup requirements along the critical paths can be removed to achieve a better circuit performance even beyond the limit of traditional sequential designs.
- The optimization with VirtualSync+ is further enhanced by fine-tuning with commercial design tools, e.g., Design Compiler from Synopsys, to achieve more accurate result. To achieve this fine-tuning, we first optimize the circuits by reallocating sequential components. Afterwards, the removal locations of flip-flops with respect to the circuits under optimization are extracted and the corresponding wave-pipelining timing constraints compatible with commercial design tools are established. These timing constraints are then incorporated into the optimization flow of commercial tools to generate the optimized circuits.

The rest of this paper is organized as follows. In Section II, we explain the motivation and the basic idea of the proposed method. The timing optimization problem is formulated in Section III. In Section IV, we provide a detailed description of the proposed timing model VirtualSync+. The proposed timing optimization with relaxed VirtualSync+ timing model is described in Section V. Circuit fine-tuning with VirtualSync+ in commercial tools is explained in Section VI. Experimental results are reported in Section VII. Conclusions are drawn in Section VIII.

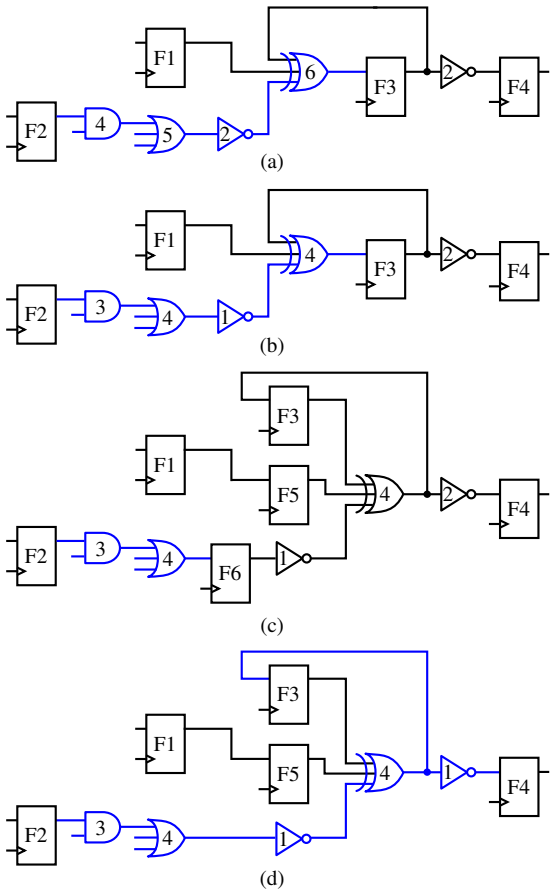


Fig. 1: Timing optimization methods. Delays of logic gates are shown on the gates. The clock-to-q delay (t_{cq}), setup time (t_{su}) and hold time (t_h) of a flip-flop are 3, 1 and 1, respectively. (a) Original circuit. (b) Sized circuit. (c) Circuit after retiming. (d) Circuit after optimization using VirtualSync+.

II. Background and Motivation

In traditional digital circuits, sequential components such as flip-flops synchronize signal propagations between pairs of flip-flops using a global clock signal, as shown in Fig. 1(a). The combinational path between F2 and F3 is critical with a path delay equal to 17. Assume that the clock-to-q delay, the setup time and the hold time of a flip-flop are 3, 1, and 1, respectively. The minimum clock period of this circuit is thus equal to 21.

To reduce the clock period, logic gates with smaller delays can be selected from the library to accelerate signal propagations on the critical paths of the circuit, at the cost of additional area overhead, leading to the circuit shown in Fig. 1(b), where the logic gates that are not on the critical path still have their

original delays for the sake of saving area. After sizing, the minimum clock period of this circuit is reduced to 16 units. To reduce the clock period further, retiming can be deployed to move F3 to the left of the XOR gate as shown in Fig. 1(c), leading to a minimum clock period equal to 11.

The circuit in Fig. 1(c) has reached the limit of timing performance in the traditional timing model, and no other method except a logic redesign can reduce the clock period further. However, this strict timing constraint can still be relaxed by removing F6 from the circuit, leading to the circuit in Fig. 1(d). If the signal from from F2 can reach the sink flip-flops F3 and F4 after the next rising clock edge and before the rising edge two periods later, data can still be latched by F3 and F4 correctly. Since the inverter before F4 can also be sized further, the largest path delay including clock-to-q delay is 16. The minimum clock period constrained by this path is thus (16+1)/2=8.5. In this scenario, the minimum clock period of the circuit is limited by the delay of the path from F5 to F4, which is 9, 18.2% lower than retiming. The circuit in Fig. 1(d) is one of the solutions with VirtualSync+, where F6 is removed and F5 as well as F3 are inserted back to block fast signals.

Since F6 can be removed from the circuit without affecting its function in fact, it makes no contribution to the logic function or timing performance in Fig. 1(c). However, the flip-flop F5 in Fig. 1(c) cannot be removed, because the signal from F1 should also arrive at F4 later than one clock period. Without F5, the signal from F1 arrives at F4 even before the next rising clock edge, and thus a loss of logic synchronization arises compared with the circuit in Fig. 1(a). Comparing Fig. 1(b) and Fig. 1(d), we can see that F3 in Fig. 1(b) blocks the fast path from F1 to F4 and breaks the feedback loop to avoid loss of logic synchronization, but it degrades the circuit performance by delaying the signal from F2 to F4 too.

The concept to allow logic signals to span several sequential stages without a flip-flop separating them is called *wave-pipelining* [23]. Previously, this technique has only been explored in the context of circuit design, where the numbers of waves on logic paths should be defined and their synchronization should be maintained by designers during the design phase. Since logic design and timing cannot be handled separately as in traditional synchronous designs, wave-pipelining becomes incompatible with the traditional fully synchronous design paradigm, which prevents its adoption in practical designs. In VirtualSync+, we introduce a new timing model that allows multiple waves on logic paths as a technique of timing optimization for sequential digital circuits in the traditional design style. The resulting circuits still provide correct timing interfaces to sequential components, e.g., flip-flops, at the boundary of the optimized circuits to maintain timing compatibility.

III. Problem Formulation

In digital circuits, we propose that the essential function of sequential components is to delay signals along fast paths in a circuit. For example, in Fig. 1(d), F5 must be kept in the circuit to delay the signal propagation from F1 to F4. The sequential components that only sit on the critical path can thus be removed to improve circuit performance, such as F6 in Fig. 1(d).

In the VirtualSync+ framework, we remove all flip-flops and then identify the necessary locations to block fast signals using combinational gates and sequential components, e.g., buffers, flip-flops, and latches. The advantage of this formulation is that it is possible to insert the minimum number of delay units into the circuit to achieve the theoretical minimum clock period.

The problem formulation of VirtualSync+ is described as follows:

Given: the netlist of a digital circuit; the delay information of the circuit; the target clock period T.

Output: a circuit with adjusted number and locations of sequential components; logic gates with new sizes; inserted delay units, e.g., buffers.

Objectives: the circuit should maintain the same function viewed from the sequential components at the boundary of the optimized circuit; the target timing specification should be met; the area of the optimized circuit should be reduced.

In the following sections, we will introduce the proposed VirtualSync+ timing optimization framework. Delay units and their insertion with a complete timing model are first explained in Section IV. Since it is time-consuming to optimize circuits with the complete model, we introduce a heuristic optimization framework with relaxed VirtualSync+ timing model in Section V. This optimization is further applied together with Design Compiler from Synopsys to achieve more accurate results in Section VI.

IV. VirtualSync+ Timing Model

A. Delay Units

In the VirtualSync+ framework, we first remove all sequential components, flip-flops, from the circuit under optimization. Consequently, logic synchronization may be lost because signals across fast paths may arrive at flip-flops in incorrect clock cycles, e.g., earlier than specified, or timing violations may be incurred. In addition, signals along combinational loops should also be blocked to avoid the loss of logic synchronization. For example, in Fig. 1(d), the combinational loop across the XOR gate must have a sequential component; otherwise a signal loses synchronization after traveling across it many times.

To slow down a signal, three different components can be used as delay units, namely, combinational gates such as buffers, flip-flops, and latches, which exhibit different delay characteristics, as shown in Fig. 2, where input/output time refers to the input/out arrival time of a signal.

In Fig. 2(a), a combinational delay unit adds the same amount of delay to any input signal. Consequently, the arrival time s_v at the output of the combinational delay unit is linear to the arrival time s_u at the input of the delay unit. Therefore, the absolute gap between the early and late arrival times of signals through short and long paths does not change when a combinational delay unit is passed through.

In delaying input signals, a flip-flop, as a sequential delay unit, behaves completely differently from a combinational delay unit, as shown in Fig. 2(b). If the arrival time of a signal falls into the time window $[t_h, T-t_{su}]$, where t_h is the hold time and t_{su} is the setup time, the output signal always leaves at the time $T+t_{cq}$, with t_{cq} as the clock-to-q delay of the

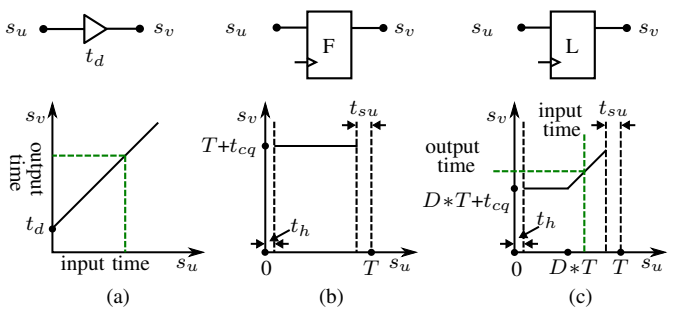


Fig. 2: Properties of delay units, assuming the launching and capturing times of an active clock edge are 0 and T, respectively. (a) Linear delaying effect of a combinational delay unit. (b) Constant delaying effect of a flip-flop. (c) Piecewise delaying effect of a latch.

flip-flop. Therefore, the gap between the early and late arrival times of two signals reaching the input of a flip-flop is always reduced to zero at the output of the flip-flop. This is a very useful property because the delays of short paths and long paths in a circuit may differ significantly after all sequential components are removed from the circuit under optimization. For many short paths, it is not possible to increase their delays by adding combinational delay units such as buffers to them, because the combinational delay units on the short paths may also appear on other long paths. The increased delays along long paths might affect circuit performance negatively. Flip-flops are thus of great use in this scenario, because short paths receive more delay padding than long paths to align logic waves in the circuit.

As the second type of sequential delay units, level-sensitive latches have a delay property combining those of combinational delay units and flip-flops, as shown in Fig. 2(c), where $0<D<1$ is the duty cycle of the clock signal. Assume that a latch is non-transparent in the first part of the clock period and transparent in the second part of the clock period. If two input signals arrive at a latch when it is non-transparent, the output gap is reduced to zero. If both signals arrive at a latch when it is transparent, the gap remains unchanged. However, if the fast signal reaches the latch when it is non-transparent while the slow signal reaches it when it is transparent, the gap between the early and late arrival times of two signals is neither zero nor unchanged. Instead, it takes a value between the two extreme cases as illustrated in Fig. 2(c). This property gives us more flexibility to modulate signals with different arrival times, specifically those along critical paths where fast signals require more delay padding and slow signals should not be affected.

B. Relative Timing References

In Fig. 1(a), if all the logic gates and flip-flop F3 are considered as the circuit under optimization, F1, F2 and F4 are thus the boundary flip-flops. No matter how signals inside the circuit propagate, the function of the whole circuit is still maintained if we can guarantee that for any input pattern at flip-flops F1 and F2 the circuit produces the same result at F4 at the same clock cycle as the original circuit.

Consider a general case in Fig. 3, where F1 and F4 are the boundary flip-flops and F2 and F3 are removed in the initial

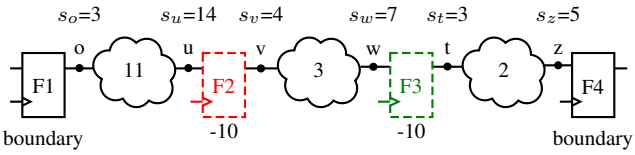


Fig. 3: Concept of relative timing references. Clock period T=10. Clock-to-q delay t_{cq}=3. Both setup time t_{su} and hold time t_h are equal to 1. F3 is kept in the optimized circuit and F2 is not included.

circuit for optimization. At F4, the arrival times are required to meet the setup and hold time constraints, written as

$$s_z+t_{su}\leq T \tag{1}$$

$$s'_z\geq t_h \tag{2}$$

where s_z and s'_z are the latest and earliest arrival times at z. These two constraints in fact are defined with respect to the rising clock edge at F3, since the clock period T in (1) shows that the signal should arrive at F4 within one clock period. Although F2 and F3 are removed from the circuit, the constraints at F4 should still be the same as (1)-(2) to maintain the compatibility of the timing interface at the boundary flip-flops.

In the general case in Fig. 3, we can also observe that the timing constraint at F3 in the original circuit is also defined with respect to the rising clock edge at F2. This definition can be chained further back until the source flip-flop F1 at the boundary is be reached. We call the locations of these removed flip-flops such as F2 and F3 ***anchor points***. After all sequential components are removed from the circuit under optimization, these anchor points still allow to relate timing information to boundary flip-flops. Every time when a signal passes an anchor point, its arrival time is converted by subtracting T in VirtualSync+. When a signal finally arrives at a boundary flip-flop along a combinational path, its arrival time must be converted so many times as the number of flip-flops on the path, so that (1)-(2) is still valid.

In Fig. 3, assume that F2 is removed but F3 is inserted back in the optimized circuit. The arrival time s_u is subtracted by the clock period T=10 to convert it with respect to the time at F1, leading to s_v=4. The arrival time s_w is defined with respect to the previous flip-flop before F3, so that the timing constraints can be checked using (1)-(2). Since the arrival time before F4 should meet its timing constraints, F3 thus cannot be removed. Otherwise, the arrival time s_t would be equal to 7-10=-3. Accordingly, the arrival time s_z becomes -3+2=-1, definitely violating the hold time constraint in (2).

Since F3 is kept in the optimized circuit, it introduces the delay with the property shown in Fig. 2(b). The arrival time after this sequential delay unit thus becomes $T+t_{cq}$=13. This signal at t in Fig. 3 also passes an anchor point. Therefore, the arrival time s_t is equal to 3, leading to no timing violation at F4. This example demonstrates that the timing constraints at the boundary flip-flops force the usage of the internal sequential delay units. The model to automatically insert these delay units will be explained in the next section.

C. Synchronizing Logic Waves by Delay Units

With all flip-flops removed from the circuit under optimization, we only need to delay signals that are so fast that they reach boundary flip-flops too early; signals that propagate slowly are already on the critical paths, thus requiring no additional delay. Since it is not straightforward to determine the locations for inserting additional delays, we formulate this task as an ILP problem and solve it later with introduced heuristic steps. The values of variables in the following sections are determined by the solver, unless they are declared as constants explicitly.

The scenario of delay insertion at a circuit node, i.e., a logic gate, is illustrated in Fig. 4, where a combinational delay unit ξ_{uv} may be inserted, the original delay of the logic gate may be sized, and a sequential delay unit may be inserted to block fast and slow signals with different delays. Furthermore, the number of flip-flops between t and z in the original circuit is represented by an integer constant λ_{tz}. When $\lambda_{tz}\geq 1$, an anchor point is found at the location between t and z. λ_{tz} is used to convert arrival times.

1) Combinational delay unit and gate sizing

In Fig. 4, the delay at the circuit node can be changed by sizing the delay of the logic gate, e.g., the XOR gate in Fig. 4. For the case that the required gate delay exceeds the largest permissible value, a combinational delay unit is inserted at the corresponding input. For convenience, we assume the combinational delay unit inserted at the input is implemented with buffers. The relation between the arrival times u and w is thus expressed as

$$s_w \geq s_u + \xi_{uv} * r^u + d_{vw} * r^u \tag{3}$$
$$s'_w \leq s'_u + \xi_{uv} * r^l + d_{vw} * r^l \tag{4}$$

where s_u, s'_u, s_w and s'_w are the latest and earliest arrival times of node u and w , respectively. ξ_{uv} is the extra delay introduced by an inserted buffer and d_{vw} is the pin-to-pin delay of the logic gate. If ξ_{uv} is reduced to 0 after optimization, no buffer is required in the optimized circuit. The $\leq$ and $\geq$ relaxations of the relation between arrival times guarantee that only the latest and the earliest arrival times from multiple inputs are propagated further. r^u and r^l are two constants to reserve a guard band for process variations, so that $r^u > 1$ and $r^l < 1$.

Rising and falling pin-to-pin delays of logic gates, e.g., the XOR gate in Fig. 4, might be different. Using the rising and falling pin-to-pin delays of logic gates to separately establish the constraints of arrival times in (3) and (4) incurs high computation complexity, and thus long execution time. For simplification, we evaluate d_{vw} with the average of the rising and falling pin-to-pin delays of the logic gate. This simplification may lead to inaccuracy in evaluating arrival times of signals and thus affect the subsequent timing optimization negatively. To compensate this inaccuracy, we calibrate the arrival times of signals with commercial tools, which will be explained in Section VI-A.

2) Insertion of sequential delay units

Since arrival times through long and short paths reaching w may have a large difference, we may need to insert sequential delay units to delay the fast signal more than the slow signal. This can be implemented with the sequential units shown in Fig. 2, where the gap between the arrival times is reduced after passing a sequential delay unit, either a flip-flop or a latch. To insert a sequential delay unit, three cases need to be examined. These cases have an "either-or" relationship, which can be converted into equivalent linear forms [31].

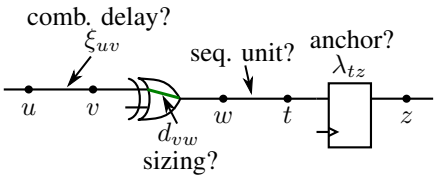


Fig. 4: Delay insertion model in VirtualSync+.

Case 1: No sequential delay unit is inserted between w and t in Fig. 4, so that

$$s_t \geq s_w \tag{5}$$
$$s'_t \leq s'_w. \tag{6}$$

Case2: A flip-flop is inserted between w and t. Assume the flip-flop works at a rising clock edge. As shown in Fig. 2(b), a flip-flop only works properly in a region t_h after the rising clock edge and t_{su} before the next rising clock edge. Therefore, we need to bound the arrival times s_w and s'_w into such a region by

$$s_w, s'_w \geq N_{wt} * T + \phi_{wt} + t_h * r^u \tag{7}$$
$$s_w, s'_w \leq (N_{wt}+1) * T + \phi_{wt} - t_{su} * r^u \tag{8}$$

where N_{wt} is an integer variable whose value is determined by the solver. This variable represents that the signal arrival time of w can be within any clock cycle, the starting and ending time of which are $N_{wt} * T$ and $(N_{wt}+1) * T$, respectively. T is the given clock period. ϕ_{wt} is phase shift of the clock signal. The available values of ϕ_{wt} can be set by designers. If only one clock signal is available, ϕ_{wt} can be set to 0 and T/2 to emulate flip-flops working at rising and falling clock edges.

When the input arrival times fall into the valid region of a flip-flop as constrained by (7)–(8), the signal always starts to propagate from the next active clock edge, so that the constraints can be written as

$$s_t \geq (N_{wt}+1)T + \phi_{wt} + t_{cq} * r^u \tag{9}$$
$$s'_t \leq (N_{wt}+1)T + \phi_{wt} + t_{cq} * r^l. \tag{10}$$

Case3: A level-sensitive latch is inserted between w and t. To be consistent with the active region of flip-flops, we assume that the latches are transparent when the clock signal is equal to 0. We can then bound the arrival times at w the same as (7)–(8).

As illustrated in Fig. 2(c), the latch is non-transparent in the first part of the region and transparent in the second region. Accordingly, the latest time a signal leaves the latch can be expressed as

$$s_t \geq N_{wt} * T + \phi_{wt} + D * T + t_{cq} * r^u \tag{11}$$
$$s_t \geq s_w + t_{dq} * r^u \tag{12}$$

where (11) corresponds to the case that the latch is non-transparent, so that the signal leaves the latch at the moment the clock switches to 1. D is the duty cycle of the clock signal with $0<D<1$. (12) corresponds to the case that the latch is transparent, so that only the delay of the latch is added to s_w. t_{dq} is the data-to-q delay of the latch.

The earliest time a signal leaves the latch is, however, imposed by a constraint in the less-than-max form as in [32],

$$s'_t \leq \max\{N_{wt}*T+\phi_{wt}+D*T+t_{cq}*r^l, s'_w+t_{dq}*r^l\} \quad (13)$$

which cannot be linearized easily. In the VirtualSync+ framework, the purpose of introducing the sequential delay unit is to delay the short path as much as possible. This effect happens when a signal arrives at a non-transparent latch. Therefore, we impose the arrival times of fast signals to be positioned in the non-transparent region, expressed as

$$N_{wt}*T+\phi_{wt}+t_h*r^u \leq s'_w \leq N_{wt}*T+\phi_{wt}+D*T \quad (14)$$

while relaxing (13) as

$$s'_t \leq N_{wt}*T+\phi_{wt}+D*T+t_{cq}*r^l. \quad (15)$$

When inserting the sequential delay unit, each of the three cases above can happen in the optimized circuit. We use an integer variable to represent the selection and let the solver determine which case happens during the optimization.

3) Reference shifting with respect to anchor points

The arrival times in the model need to be converted each time when an anchor point is passed. The constant λ_{tz} represents the number of flip-flops at such a point in the original circuit. In Fig. 4, the arrival time at z is shifted as

$$s_z = s_t - \lambda_{tz} T. \quad (16)$$

4) Wave non-interference condition

Since we allow multiple waves to propagate along a combinational path, we need to guarantee that the signal of the next wave starting from a boundary flip-flop never catches the signal of the previous wave starting from the same flip-flop [23]. This constraint should be imposed to every node in the circuit. For example, the constraint for node u is written as

$$s_u + t_{stable} \leq s'_u + T \quad (17)$$

where t_{stable} is the minimum gap between two consecutive signals.

5) Overall formulation

The introduction of the relative timing references, or the anchor points, in Section IV-B guarantees that the number of clock cycles along any path does not change after optimization. With the timing constraints (1)-(2) at boundary flip-flops, the correct function of the optimized circuit is always maintained, without requiring any change in other function blocks.

The constraints (1)–(17) excluding (13) need to be established at each node in the circuit after flip-flops are removed to guarantee the correct timing interface to flip-flops at the boundary of the optimized circuit parts. If this timing interface is correct, the functionality of the circuit is maintained [23], [25], [28]. The appearance of the combinational and sequential

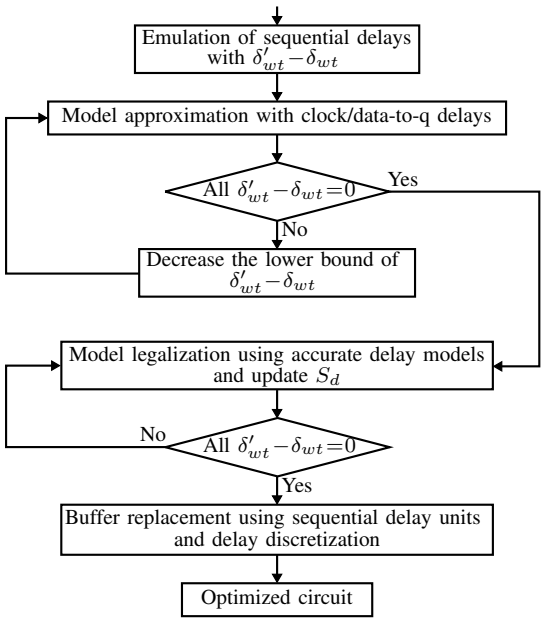


Fig. 5: The proposed timing optimization with relaxed VirtualSync+ flow.

delay units needs to be determined by the solver. The delays of logic gates should also be sized. The objective of the optimization is to find a solution to make the circuit work at a given clock period T, while reducing the area cost. Taking all these factors into account, the straightforward ILP formulation may become insolvable. In practice, however, this technique only needs to be applied to isolated circuit parts containing critical paths. In addition, we introduce heuristic techniques to overcome this scalability problem, as explained in the following section.

V. Timing Optimization with Relaxed VirtualSync+ Timing Model

In applying the VirtualSync+ timing model above, we introduce a framework to identify the locations of delay unit with iterative relaxation of VirtualSync+. The flow of this framework is shown in Fig. 5, which will be explained in this section. The basic strategy is to remove flip-flops along critical paths to eliminate the inherent clock-to-q delays and setup time. Thereafter, fast signals of short paths will be blocked to guarantee the correct functionality by inserting the minimum number of sequential delay units and buffers. To reduce area overhead, buffers are replaced with sequential delay units.

A. Emulation of Sequential Delay Units

After we remove all the flip-flops from the original circuit, the short paths may have extremely small delays. The gap between these delays and those of long paths is very large. It cannot be reduced with combinational delay units since they introduce the same delays to the fast and slow signals as shown in Fig. 2(a). Instead, only sequential delay units are able to reduce this gap so that the fast and slow signals still arrive at boundary flip-flops within the same clock cycle as those of the original circuit.

In the first step of the framework, we identify the locations at which sequential delay units are indispensable. Without

these units, the fast and slow signals, such as those within feedback loops, may not be aligned properly into the correct clock cycles, even though unlimited combinational delay units can be inserted into the circuit. In practice, however, it is not easy to identify the exact locations of these units using the exact and complete model in (5)–(15) directly. To solve this problem, we relax the timing constraints (5)–(15) instead of modeling them directly. From Fig. 2, we can see that a buffer slows down fast and slow signals with the same delay. On the contrary, a sequential delay unit slows down fast and slow signals with different delays. For example, no matter when fast and slow signals arrive at the input of a flip-flop, they leave the output of the flip-flop at the same time. Accordingly, the flip-flop slows down the fast signal with a larger delay than the slow signal. Taking advantage of this characteristic, we use two non-negative variables δ_{wt} and δ'_{wt} to emulate the delay effects of three delay units as described above. In case $\delta_{wt}=\delta'_{wt}$, a buffer is inserted. In case $\delta_{wt}<\delta'_{wt}$, a sequential delay unit is inserted. When signals travel from u to z in Fig. 4, the relation of arrival times from nodes u to z can be written as

$$s_z \geq s_u+\xi_{uv}*r^u+d_{vw}*r^u+\delta_{wt}-\lambda_{tz}T \tag{18}$$

$$s'_z \leq s'_u+\xi_{uv}*r^l+d_{vw}*r^l+\delta'_{wt}-\lambda_{tz}T \tag{19}$$

$$0 \leq \delta_{wt} \leq \delta'_{wt} \tag{20}$$

$$s'_u+\delta'_{wt} \leq s_u+\delta_{wt} \tag{21}$$

where the variables δ_{wt} and δ'_{wt} emulate delays introduced by sequential delay units. (20) specifies that the fast signal should be padded with more delays than the slow signal. The purpose of (21) is to guarantee that a sequential delay unit is required to break a feedback loop.

The optimization problem to find the potential locations of sequential delay units is thus written as

$$\text{minimize} \quad \alpha\sum_{G}(\delta'_{wt}-\delta_{wt})+\beta\sum_{G}(\delta'_{wt}+\xi_{uv})-\gamma\sum_{G}d_{vw} \tag{22}$$

$$\text{subject to} \quad \text{(17)–(21) for each gate in } G \tag{23}$$

$$\text{constr. (1)–(2) for each boundary flip-flop} \tag{24}$$

where G is the set of all logic gates in the original circuit. This optimization problem also maximizes the overall delays of logic gates in the circuit with the last term in (22) since larger delays indicate smaller area. Since the area of a flip-flop is about 6 times of the area of a buffer and the average area of combinational gates, α, β and γ are set to 100,10,10, to specify the balance between the area of sequential delay units, inserted buffers and logic gates. With this setting, we provide the solver a clear tendency to minimize the area of sequential delay units, buffers and combinational gates with different priorities. In the second term, δ'_{wt} also represents the existence of buffers in case $\delta'_{wt}=\delta_{wt}$. Solving the optimization problem above identifies nodes with unequal padding delays δ_{wt} and δ'_{wt}, indicating potential locations of sequential delay units, as a set S. These delays may still violate the exact constraints in (5)–(15), so that they need to be refined further.

B. Modeling with Clock/Data-to-Q Delays of Sequential Delay Units

The optimization problem (22)–(24) does not consider the inherent clock-to-q delays of flip-flops and data-to-q delays of latches. Since these delays are introduced only at locations where sequential delay units are inserted, they need to be modeled for all the locations S returned by the previous step. We introduce a binary variable x_{wt} to represent whether a sequential delay unit appears at a location from S, and revise the constraints (18)–(19) as

$$s_z \geq s_u+\xi_{uv}*r^u+d_{vw}*r^u+x_{wt}\delta_{wt}+x_{wt}t_{cd\to q}*r^u-\lambda_{tz}T \tag{25}$$

$$s'_z \leq s'_u+\xi_{uv}*r^l+d_{vw}*r^l+x_{wt}\delta'_{wt}+x_{wt}t_{cd\to q}*r^l-\lambda_{tz}T \tag{26}$$

where $t_{cd\to q}$ represents clock-to-q delay or data-to-q delay, which can be evaluated according to the output load of the sequential components, and the delays δ_{wt}, δ'_{wt} and $t_{cd\to q}$ are only valid when x_{wt} is equal to 1. The inclusion of the binary variables x_{wt} is very computation-intensive, so that they can only be dealt with after the potential locations of sequential delay units are reduced to S by solving (22)–(24). Since x_{wt} is a binary variable, the multiplications $x_{wt}\delta_{wt}$ and $x_{wt}\delta'_{wt}$ can be converted into equivalent linear forms so that the overall formulation is still an ILP problem.

Considering the inherent delays of sequential delay units, their locations can be refined further by solving the optimization problem as

$$\text{minimize} \quad \alpha\sum_{G/S}(\delta'_{wt}-\delta_{wt})+\beta\sum_{G/S}(\delta'_{wt}+\xi_{uv})-\gamma\sum_{G}d_{vw} \tag{27}$$

$$\text{subject to} \quad \text{(17)–(21) for each gate in } G\backslash S \tag{28}$$

$$\text{(17), (20)–(21) for each gate in } S \tag{29}$$

$$\text{(25)–(26) for each gate in } S \tag{30}$$

$$\text{constr. (1)–(2) for each boundary flip-flop} \tag{31}$$

$$\delta'_{wt}-\delta_{wt} \geq d_{th} \text{ if } x_{wt}=1 \tag{32}$$

where d_{th} is a predefined delay bound. The condition in (32) can be converted into equivalent linear forms according to [31].

To identify the necessary locations of sequential delay units, we execute the optimization (27)–(32) iteratively until no different δ_{wt} and δ'_{wt} exist anymore. In each iteration, we lower the delay bound d_{th} linearly. A large bound of $\delta'_{wt}-\delta_{wt}$ in the early iterations allows the solver to quickly determine the important locations for inserting sequential delay units. The refined locations of sequential delay units from this step are returned as a set S_d, which is more accurate than S.

C. Model Legalization for Timing of Sequential Delay Units

In this step, the complete model described in Section IV-C is applied to the locations in S_d to generate sequential delay units that are really required in the circuit. The optimization

problem is described as

$$\text{minimize} \quad \alpha \sum_{G/S_d} (\delta'_{wt} - \delta_{wt}) + \beta \sum_{G/S_d} (\delta'_{wt} + \xi_{uv}) - \gamma \sum_{G} d_{vw} \tag{33}$$

$$\text{subject to} \quad \text{(17)–(21) for each gate in } G \backslash S_d \tag{34}$$

$$\text{(5)–(12) for each gate in } S_d \tag{35}$$

$$\text{(14)–(17) for each gate in } S_d \tag{36}$$

$$\text{constr. (1)–(2) for each boundary flip-flop.} \tag{37}$$

After solving the optimization above, there might still be different δ_{wt} and δ'_{wt} in $G \backslash S_d$, because the timing legalization of sequential delay units with the complete model (1)–(17) excluding (13) in Section IV-C may invalidate some locations in S_d so that S_d reduces in size. The complete model is applied to these locations iteratively, until no different δ_{wt} and δ'_{wt} exists, indicating the remaining timing synchronization can be achieved with buffers and gate sizing directly.

D. Buffer Replacement with Sequential Units

After solving (33)–(37), delays d_{vw} of logic gates are set to the nearest discrete delay values defined in the library. Buffer delays ξ_{uv} are also determined. If ξ_{uv} is large, several buffers are needed for its implementation. As shown in Fig. 2, sequential delay units can introduce a very large delay. For example, a flip-flop can introduce a delay as large as $T + t_{cq} - t_h$, if the incoming signal arrives at the flip-flop right after a clock edge. According to this observation, we iteratively replace buffers with large delays by sequential delay units to reduce area. In each iteration, the accurate sequential model (5)–(12) and (14)–(17) is applied to guarantee these new sequential delay units are valid. The iteration stops when no buffer can be replaced by sequential units. Buffers that cannot be replaced by sequential delay units are implemented directly in the optimized circuit.

VI. Circuit Fine-Tuning with VirtualSync+ in Commercial Tools

The timing optimization with relaxed VirtualSync+ described in Section V can generate the optimized circuits where timing performance is improved and area cost is reduced. However, this method might be inconsistent when it is interfaced with commercial tools, e.g., Design Compiler from Synopsys. For example, the average pin-to-pin delays are used in evaluating the arrival times of signals for the sake of execution efficiency in the relaxed VirtualSync+. Therefore, we fine-tune the optimized circuits with Design Compiler.

To fine-tune the optimized circuits with Design Compiler, there are several challenges. First, the arrival times of signals should be evaluated accurately to make the optimization effective. Second, after the optimization, the removal locations of flip-flops in the optimized circuits should be extracted to establish the wave-pipelining timing constraints, with which Design Compiler can fine-tune the circuits. Third, buffers should be adjusted along short paths to satisfy the wave-pipelining timing constraints.

To overcome the challenges described above, we first adjust the iterative relaxation method where iterations of delay unit insertion are executed, as described in Section V, to compensate the inaccuracy in timing optimization. Afterwards, we extract the removal locations of flip-flops with respect to the circuits under optimization and establish the corresponding wave-pipelining constraints compatible with Design Compiler. These timing constraints are then incorporated into the commercial tools to optimize the circuits. After this optimization, the delays of short paths might be still too small to meet the wave-pipelining constraints, so that buffers are inserted to pad their delays until timing constraints are met. These steps can be automatically executed without manual intervention.

A. Adjusting Iterative Relaxation for Fine-Tuning of VirtualSync+

During the iterative relaxation described in Section V, the average pin-to-pin delays are used in evaluating the arrival times of signals for the sake of execution efficiency. However, this simplification might incur inaccuracy in evaluating the arrival times of signals and thus path delays. Therefore, critical paths might be considered as non-critical and thus cannot be optimized effectively. To solve this problem, we adjust the iterative relaxation in Section V to compensate this evaluation inaccuracy in timing optimization. In this adjustment, we first run timing slack analysis for all the flip-flops of a circuit with Design Compiler before executing the iterative relaxation framework. Afterwards, the latest and earliest arrival times of signals at the inputs of flip-flops evaluated with Design Compiler are used to calibrate those evaluated with the average pin-to-pin delays. Specifically, we record the gaps between arrival times evaluated with Design Compiler and with average pin-to-pin delays. During the optimization with the iterative relaxation, the arrival times at the inputs of flip-flops evaluated with average pin-to-pin delays are calibrated by adding the corresponding gaps. With this calibration, the critical paths can be identified and optimized effectively.

During the timing optimization with the iterative relaxation framework, gate sizing plays an important role of increasing/decreasing delays for short/long paths to meet the timing constraints. Since Design Compiler has the advantages of high accuracy and execution efficiency in sizing gates while minimizing the area overhead, we shift the task of gate sizing to Design Compiler in the adjusted iterative relaxation framework. Specifically, we first adopt the adjusted iterative relaxation without gate sizing to insert necessary sequential delay units to block fast signals along short paths in a circuit. Afterwards, we use Design Compiler to size gates to increase/decrease delays of short/long paths. If the delays of short paths are still too small after gate sizing, buffers are inserted to pad their delays, as explained later.

B. Extracting Removal Locations of Flip-flops

After the adjusted iterative relaxation is executed, flip-flops along critical paths are removed and fast signals are blocked with sequential delay units in the optimized circuit. Therefore, the sequential components in the optimized circuit are reallocated with respect to the circuit under optimization. Fig. 6 illustrates a circuit under optimization and the optimized circuit where red dots represent anchor points, also the removal locations of flip-flops. By comparing the circuit under

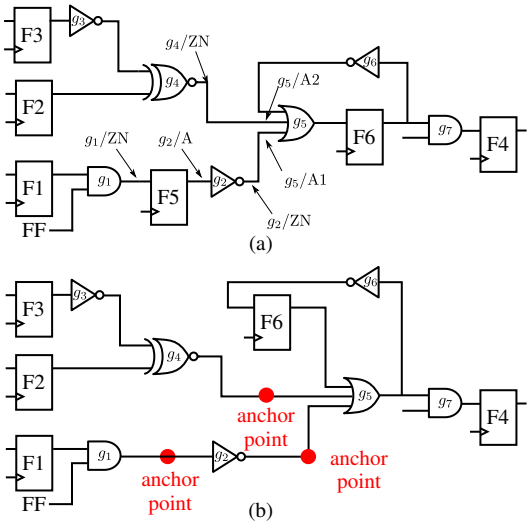


Fig. 6: The circuit under optimization and the optimized circuit after the adjusted iterative relaxation. (a) The circuit under optimization where g_i/ZN represent the output pin of g_i and g_i/A, g_i/A1 and g_i/A2 are the input pins of g_i. (b) the optimized circuit where flip-flops are removed.

optimization in Fig. 6(a) and the optimized circuit in Fig. 6(b), we can see that F6 in the circuit under optimization is moved leftwards with retiming to block signals along a loop, while the retimed flip-flops between g_4 and g_5, g_2 and g_5 and F5 are removed to allow slow signals to propagate through.

The removal of flip-flops creates wave-pipelining paths where several logic waves propagate simultaneously in the optimized circuit. For example, in Fig. 6(b), the wave-pipelining path from F2 to F4 has two logic waves propagating along it since this path traverses through one removal location along it and the wave-pipelining path from F1 to F4 has three logic waves propagating along it since it traverses through two removal locations along it. To guarantee the correct functionality of the optimized circuit, the delays of such paths should satisfy the corresponding wave-pipelining timing constraints. For example, the largest delay of the path between F1 and F4 should be smaller than $3{\cdot}T{-}t_{su}$, and the smallest delay should be larger than $2{\cdot}T{+}t_h$, where T is the target clock period, t_{su} and t_h are the setup time and hold time of F4, respectively. To optimize such paths to satisfy the wave-pipelining constraints, we should establish the wave-pipelining constraints compatible with Design Compiler and then incorporate such constraints into the optimization flow of Design Compiler. To achieve this goal, we first extract the removal locations of flip-flops with respect to the circuit under optimization.

According to Fig. 6, the removal locations of flip-flops in the optimized circuit can be extracted by applying retiming together with removing flip-flops on the circuit under optimization. To apply this technique on the circuit under optimization, we use $g{\in}G$ to represent a combinational gate and $e_{g_i,g_j}{\in}E$ between gates g_i and g_j to represent the net connecting the output of the combinational gate g_i and an input of another combinational gate g_j. e_{g_i,g_j} has a constant weight $w(e_{g_i,g_j})$ to represent the number of flip-flops along the connection in the circuit under optimization. Each combinational gate has a retiming variable $r(g)$, which defines how many flip-flops are moved from the output of a gate to its inputs. After retiming, the number of flip-flops on a net between gates g_i and g_j is written as $w_r(e_{g_i,g_j}){=}w(e_{g_i,g_j}){+}r(g_j){-}r(g_i)$. The number of flip-flops along e_{g_i,g_j} in the optimized circuit is denoted as $w'(e_{g_i,g_j})$, which can be different from $w_r(e_{g_i,g_j})$ due to the removal of flip-flops. To determine how many flip-flops along e_{g_i,g_j} are removed in the optimized circuit, a variable $y_{e_{g_i,g_j}}$ is assigned for e_{g_i,g_j}. With this setting, two cases for a net between gates g_i and g_j should be examined.

Case 1: If the net from gate g_i to gate g_j has the retimed weight equal to the weight in the optimized circuit, namely $w_r(e_{g_i,g_j}){=}w(e_{g_i,g_j}){+}r(g_j){-}r(g_i){=}w'(e_{g_i,g_j})$, there is no removal of flip-flops along this net, so that $y_{e_{g_i,g_j}}{=}0$. For example, $y_{e_{g_5,g_6}}{=}0$ in case of $r(g5){=}1$.

Case 2: If the net from gate g_i to gate g_j has the retimed weight larger than that in the optimized circuit, namely $w_r(e_{g_i,g_j}){=}w(e_{g_i,g_j}){+}r(g_j){-}r(g_i){\geq}w'(e_{g_i,g_j}){+}1$, flip-flops are removed along this net, and the number of removed flip-flops is $y_{e_{g_i,g_j}}{=}w_r(e_{g_i,g_j}){-}w'(e_{g_i,g_j})$. For example, $y_{e_{g_4,g_5}}{=}1$, $y_{e_{g_2,g_5}}{=}1$ and $y_{e_{g_1,g_2}}{=}1$.

When establishing the relation between the circuit under optimization and the optimized circuit, each of the cases above can happen. We use the constraints in the two cases described above to establish an ILP formulation and let the solver determine which case actually happens during the adjusted iterative relaxation. After that, we can obtain the removal locations of flip-flops along net connections where $y_{e_{g_i,g_j}}{\geq}1$ in the optimized circuit.

C. Generating Wave-pipelining Timing Constraints for Integration of VirtualSync+

After the removal locations of flip-flops are extracted with the method in Section VI-B, we can use them to automatically establish the wave-pipelining constraints compatible with Design Compiler. In Design Compiler, wave-pipelining timing constraints can be set with the following commands

$$set\_max\_delay\ d\ \text{-}from\ p_1\ \text{-}through\ p_2\ \text{...}\ \text{-}through\ p_n\ \text{-}to\ p_{n+1}$$

$$set\_min\_delay\ d'\ \text{-}from\ p_1\ \text{-}through\ p_2\ \text{...}\ \text{-}through\ p_n\ \text{-}to\ p_{n+1}$$

where "set_max_delay" and "set_min_delay" are used to establish setup and hold time constraints for long paths and short paths, respectively, "$through$" means propagating through, "$from$" and "to" mean leaving from and arriving at, d and d' are delay values and p_i is a pin name. "-$from\ p_1$" and "-$to\ p_{n+1}$" can be removed in the constraints as long as the timing constraints are sufficient to identify required paths. In these timing constraints, setup time and hold time of flip-flops are subtracted and added automatically.

In Fig. 6, the path from F1 to F4 has three logic waves since it traverses through two removal locations along it, namely the connection between $g1$/ZN and $g2$/A as well as $g2$/ZN and g_5/A1. The wave-pipelining constraints from F1 to F4 can be written as follows

$$set\_max\_delay\ 3{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_2\text{/A}\ \text{-}through\ g2\text{/ZN -through } g_5\text{/A1} \quad (38)$$

$$set\_min\_delay\ 2{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_2\text{/A}\ \text{-}through\ g2\text{/ZN -through } g_5\text{/A1}. \quad (39)$$

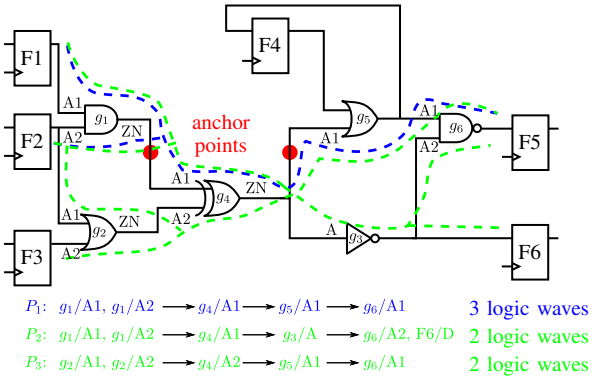


Fig. 7: Wave-pipelining paths with different number of logic waves are entangled with each other. Red dots represent anchor points, also the removal locations of flip-flops. $P1$, $P2$ and P_3 are the path set. The paths in P_1, as shown in blue, have three logic waves propagating along them and the paths in P_2 and P_3, as shown in green, have two logic waves propagating along them.

The timing constraints of the paths with two logic waves in Fig. 6, such as the paths from F2 to F4 and from F3 to F4 can be written similarly.

In Fig. 6, the wave-pipelining paths with different number of logic waves are clearly separated. However, in some optimized circuits, wave-pipelining paths with different number of logic waves are entangled with each other. Fig. 7 illustrates such a circuit, where the paths in P_1, as shown in blue color, has three logic waves propagating along it, since they traverse through two anchor points. The wave-pipelining constraints for such paths can be established with the two anchor points directly, similar to (38)-(39). For example, the setup time constraint of such paths can be written as follows

$$set\_max\_delay\ 3{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1}\ \text{-}through\ g4\text{/ZN}\ \text{-}through\ g_5\text{/A1} \quad (40)$$

However, the paths in P_2 and P_3, as shown in green color, propagate through only one of the anchor points and entangle with the blue paths with three logic waves. Directly using one of the anchor points to establish the wave-pipelining constraints for such paths with two logic waves leads to a conflict with the wave-pipelining constraints established with two anchor points. For example, if the setup constraint for the paths in P_2 is written as follow

$$set\_max\_delay\ 2{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1} \quad (41)$$

this constraint and the constraint (40) confuse Design Compiler, since the paths propagating through two anchor points also traverse through one of the anchor points. In this case, Design Compiler ignores the constraint in (41), so that the delays of the paths in P_2 cannot be optimized correctly. To solve this problem, the paths with two logic waves should be differentiated from the paths with three logic waves when establishing the corresponding timing constraints for the subsequent optimization.

To differentiate the wave-pipelining paths with two logic waves from those with three logic waves, we propose to identify the pins that separate such paths. We call such pins **differentiating pins**. For example, to differentiate the wave-pipelining paths in P_2 from those in P_1, we first find the sink flip-flops at which the paths in P_2 and P_1 arrive. As shown in Fig. 7, F5 and F6 are such sink flip-flops. Since F6 is the sink flip-flop at which only the paths in P_2 can arrive, the wave-pipelining constraints in such a case can be established with the anchor point between g_1/ZN and g_4/A1 together with the sink flip-flop as follows

$$set\_max\_delay\ 2{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1}\ \text{-}to\ \text{F6/D}$$

$$set\_min\_delay\ 1{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1}\ \text{-}to\ \text{F6/D}.$$

Since F5 is the sink flip-flop at which both wave-pipelining paths in P_1 and P_2 can arrive, we need to find the differentiating pins to separate them. To achieve this goal, we enumerate the paths starting from F5 and ending at the two anchor points backwards. For example, the path propagating through $g6$/A2, g_3/A, and g_4/A1 terminates at the anchor point between g_1/ZN and g_4/A1 and the path propagating through $g6$/A1 and $g5$/A1 terminates at the anchor point between g_4/ZN and g_5/A1. By comparing these paths, it is clear that the pin g_6/A2 differentiates the wave-pipelining paths in P_2 from those in P_1. Therefore, the wave-pipelining constraints for the paths in P_2 can be established with the differentiating pin g_6/A2 and the anchor point between g_1/ZN and g_4/A1 as follows

$$set\_max\_delay\ 2{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1}\ \text{-}through\ g_6\text{/A2}$$

$$set\_min\_delay\ 1{\cdot}T\ \text{-}through\ g_1\text{/ZN}\ \text{-}through\ g_4\text{/A1}\ \text{-}through\ g_6\text{/A2}.$$

Similarly, to differentiate the wave-pipelining paths in P_3 from those in P_1, we first find the common source flip-flops which the paths in P_3 and P_1 start from. As shown in Fig. 7, this common source flip-flop is F2. Afterwards, we enumerate the paths starting from F2 and ending at the two anchor points in a forward way. By comparing these paths, we can see that the pin g_2/A1 differentiates the wave-pipelining paths in P_3 from those in P_1. We then use the differentiating pin g_2/A1 and the anchor point between g_4/ZN and g_5/A1 to establish the wave-pipelining constraints. In a general case where N anchor points exist in an optimized circuit after the removal locations of flip-flops are extracted, we use the methods described above to establish the wave-pipelining constraints for the paths where the number of logic waves propagating along them is larger than 1. The generation of wave-pipelining constraints is fully automated without manual intervention.

D. Circuit Optimizing and Buffer Insertion with Commercial Tools

After the wave-pipelining constraints are established for the optimized circuit, we incorporate such constraints into Design Compiler to optimize this circuit. During the optimization, Design Compiler sizes gates to increase the delays in short paths to satisfy the wave-pipelining constraints. After this optimization, the delays of the short paths might still be too small to satisfy the wave-pipelining constraints. To pad their delays with Design Compiler, we insert buffers in the optimized circuits to enlarge their delays with the command "insert_buffer p -no_of_cells d t", where p is a pin name, d is the number of buffers and t is the type of buffers, e.g.,

TABLE I: Results of VirtualSync+

Circuit				Ret.&Siz.				Cri. Part		Opt. Circuit			Comparison		Runtime
	n_s	n_c	$T(ps)$	n_s'	n_c'	n_t'	n_a'	n_{cs}	n_{cc}	n_f	n_l	n_b	n_t	n_a	$t(s)$
systemcdes	574	1286	1132	523	2429	55.5%	23.1%	195	2290	65	66	31	2.5%	-5.4%	2619.5
tv80	1014	2998	1025	1228	5818	40.0%	50.4%	296	5758	158	141	383	2%	0.6%	3081.5
wb_dma	720	1632	666	716	1974	31.5%	11.4%	129	1375	65	46	240	8.5%	1.5%	2211.3
systemcaes	1181	3318	921	1635	5078	29.5%	38.0%	363	4531	258	150	141	3%	-1.5%	2052.1
mem_ctrl	879	1840	1105	1429	3318	51.0%	68.3%	290	2642	156	93	296	4.5%	-1.5%	4510.5
usb_funct	2699	4952	1042	3005	8901	35.5%	28.6%	333	6293	253	71	252	3%	0.1%	2166.5
ac97_ctrl	1145	1467	802	2021	2363	48.5%	70.9%	579	1978	430	125	163	1%	-2.2%	1961.2
pci_bridge	2301	4993	1141	6243	6476	33.5%	130.1%	264	4410	156	48	140	1%	-0.7%	3563.1

BUF_X1. The locations of buffers can be obtained from the adjusted iterative relaxation. Due to the heuristic optimization in the adjusted iterative relaxation, the number of buffers might not be large or small enough to satisfy the wave-pipelining constraints. To solve this problem, we iteratively insert/remove buffers along the short/long paths until the wave-pipelining constraints are met.

VII. Experimental Results

A. Experimental Setup

The proposed method was implemented in C++ and tested using a 3.20 GHz CPU. In the experiments, the circuits from the TAU 2013 variation-aware timing analysis contest are first optimized using 45 nm library with Design Compiler to reduce area cost. These circuits are referred to the original circuits. The clock periods of these original circuits are shown as T in the fourth column. We optimize these circuits further using the proposed VirtualSync+ to improve timing performance and reduce area. The optimization results are shown in Table I. The number of flip-flops and the number of combinational components of the original circuits are shown in the columns $\boldsymbol{n_s}$ and $\boldsymbol{n_c}$, respectively. To tolerate process variations, 10% of timing margin was assigned, so that r^u and r^l in Section V were set to 1.1 and 0.9, respectively. In case of large variations in advanced technologies, a large timing margin can be assigned by increasing r^u and decreasing r^l, respectively. The allowed phase shifts ϕ_{wt} in Section V are 0, T/4, T/2 and 3T/4. t_{stable} in equation (17) in Section IV is set to the delay of a buffer to isolate signals on the next wave and the previous wave. The initial delay bound d_{th} in Section V is 7T/8 and iteratively reduced by T/8. The ILP solver used in the VirtualSync+ framework was Gurobi.

The goal of VirtualSync+ is to enhance timing performance and thus solve timing violations in the circuits which have already been optimized. Therefore, the original circuits are first retimed and sized extremely with Design Compiler to achieve the limit of timing performance. Accordingly, extreme retiming and sizing is an integral part of the VirtualSync+ framework. This extreme retiming and sizing, abbreviated as extreme R&S, is realized by iteratively reducing the clock period and optimizing the circuits with retiming and sizing in Design Compiler until the circuits cannot be optimized further anymore. In the circuits after extreme R&S, the number of flip-flops and the number of combinational components, shown as $\boldsymbol{n_s'}$ and $\boldsymbol{n_c'}$ in Table I, are different from those in the original circuits. $\boldsymbol{n_t'}$ and $\boldsymbol{n_c'}$ show the clock period reduction and the area increase of the circuits after extreme R&S with respect to the original circuits. Overall, the timing performance of a circuit is improved significantly by extreme R&S while the area is increased.

To increase the timing performance with VirtualSync+, in the circuits after extreme R&S, combinational paths whose delays are larger than a specified clock period, which is evaluated by iteratively reducing the clock period obtained from extreme R&S, were selected. The source and sink flip-flops of these selected paths were allowed to be removed, while the other flip-flops in the circuits were considered as boundary flip-flops. All the combinational components that can reach the flip-flops at the sources or sinks of these selected paths through combinational paths are considered as the critical part of a circuit together. The extracted critical part of the circuits occupied a large portion of the circuits after extreme R&S, as shown in $\boldsymbol{n_{cs}}$ and $\boldsymbol{n_{cc}}$, which represent the percentage of the number of flip-flops and combinational components of the critical part in the circuits after extreme R&S. From $\boldsymbol{n_{cs}}$ and $\boldsymbol{n_{cc}}$, we can see that more than 4% of flip-flops and more than 68% of combinational components have been selected for timing optimization. In case of industrial designs which contain hundreds of thousands of gates in the extracted critical subcircuit, the critical subcircuit can be processed with VirtualSync+ by iteratively selecting a small part for timing optimization.

B. Experimental Results with Timing Optimization Using Relaxed VirtualSync+ Timing Model

To verify the improvement of circuit performance, we gradually reduced the clock period by 0.5% of the clock period obtained from extreme R&S and applied the timing optimization with relaxed VirtualSync+ in Section V to meet the timing constraints. The results correspond to the results described in the conference version [1] of this paper. The column $\boldsymbol{n_f}$ and $\boldsymbol{n_l}$ show the numbers of flip-flops and latches in the critical part after optimization with relaxed VirtualSync+, respectively. The sums of these numbers are comparable or even smaller than the numbers of flip-flops before relaxed VirtualSync+. The numbers of extra inserted buffers to match arrival times are shown in the column $\boldsymbol{n_b}$. Thanks to the buffer replacement with sequential units in the proposed framework, the numbers of extra inserted are not large. Compared with the number of original combinational components shown in the column n_{cc}, these numbers show that the cost due to the inserted buffers is still acceptable.

The column $\boldsymbol{n_t}$ in Table I shows the final clock period reduction compared with the circuits after extreme R&S. The

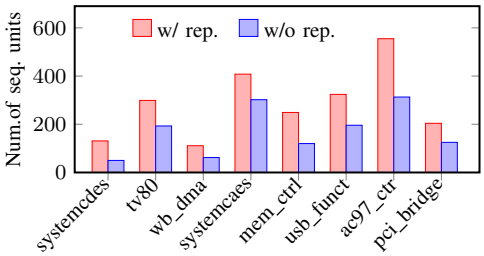


Fig. 8: Comparison of sequential delay units after buffer replacement.

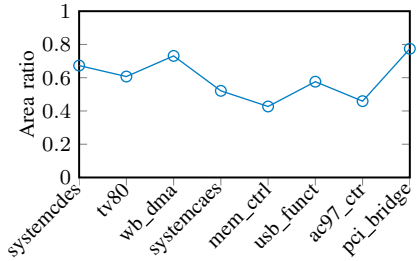


Fig. 9: Area comparison before and after buffer replacement.

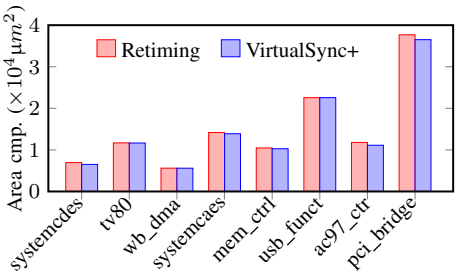


Fig. 10: Area comparisons with extreme R&S with the same clock period.

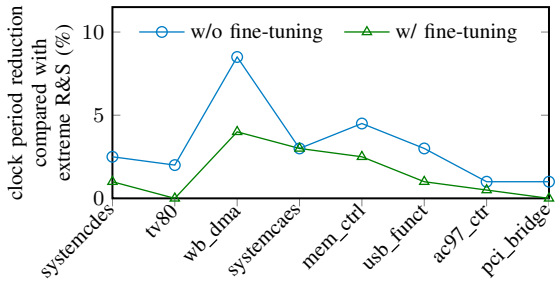


Fig. 11: Comparison of the clock period reduction before and after fine-tuning with Design Compiler.

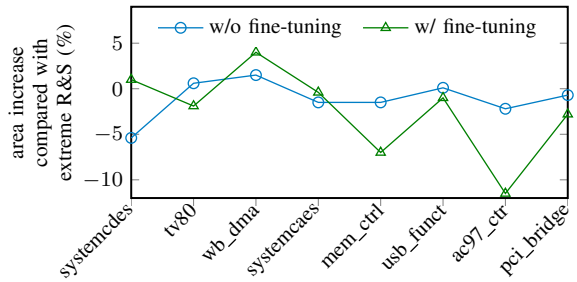


Fig. 12: Comparison of the area increase before and after fine-tuning with Design Compiler.

maximum and average reduction are 8.5% and 3.2%, respectively, which resulted from the compensation between flip-flop stages and the removal of clock-to-q delays and setup time requirements on critical paths. For all the cases, the minimum clock periods have been pushed even further than those from extreme R&S. The timing performance improvement $\boldsymbol{n_t}$ is achieved with relaxed VirtualSync+ described in Section V. This method can be used to quickly evaluate the benefit of VirtualSync+ in circuit design.

The area increase of the proposed method compared with extreme R&S is shown in column $\boldsymbol{n_a}$. In the cases with area increase, the overhead is still negligible; in other cases, the area is even smaller because unnecessary flip-flops were removed in the proposed framework. The last column t_r in Table I shows the runtime of the proposed method. Since the ILP formulation with the complete model in Section IV-C is NP-hard, it is impractical to find a solution with respect to area and clock period. In the experiments, the runtime with iterative relaxations is acceptable in remedying remaining timing violations for late design stage.

In the proposed framework, sequential delay units are first inserted only at necessary locations to delay signal propagation. Afterwards, more of them are used to replace buffers to reduce area, as described in Section V. To demonstrate

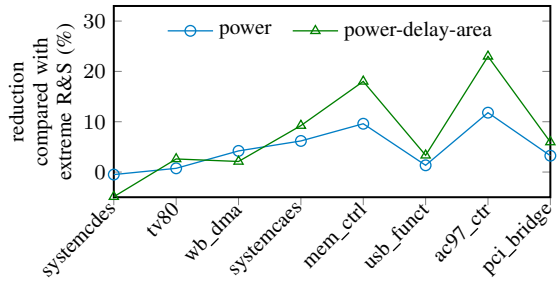


Fig. 13: The reduction of power and power-delay-area product compared with extreme R&S.

the effectiveness of the buffer replacement with sequential units in the proposed framework, we compare the numbers of sequential delay units without and with buffer replacement, as shown in Figure 8. This figure shows an increase in the number of such delay units replacing buffers. Figure 9 shows the ratio of the chip area after buffer replacement to that before buffer replacement. In all test cases, the area after buffer replacement is smaller than that before buffer replacement, demonstrating the efficiency of sequential delay units in delaying fast signals.

The comparison of the area overhead, shown as $\boldsymbol{n_a}$ in Table I, is between the clock period achieved by extreme R&S and the clock period reduced further by the proposed method. To demonstrate the area efficiency of the proposed method, we also compared the proposed method and the extreme R&S with the same clock period from the latter. The results are shown in Fig. 10. In most cases, our framework can further reduce the area achieved by extreme R&S.

C. Experimental Results with Circuit Fine-Tuning in Design Compiler

The experimental results described in Section VII-B are generated with the timing optimization using relaxed VirtualSync+ in Section V. These intermediate results might be inconsistent with commercial tools due to the simplified model. To provide more consistent results, we fine-tune the optimized circuits with VirtualSync+ in Design Compiler, as described in Section VI. After this fine-tuning, we reevaluated the improvement of timing performance and area reduction with respect to the circuits after extreme R&S. The comparisons between the clock period reduction and the area reduction with and without the fine-tuning with Design Compiler are illustrated in Fig. 11 and Fig. 12, respectively. According to these two figures, it is clear that in most cases the timing performance with the fine-tuning can still be pushed beyond that after extreme R&S while the area is reduced. Specifically, the timing performance

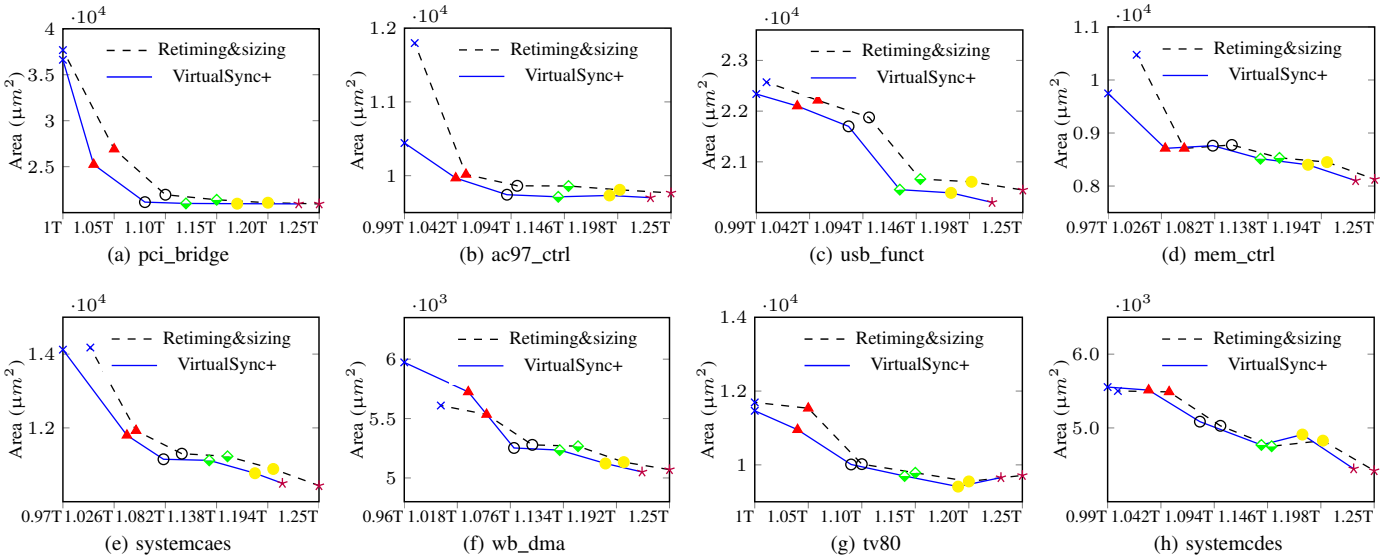


Fig. 14: Area comparison between relaxed retiming and sizing and VirtualSync+ when the target clock period is relaxed to 1.05T, 1.10T, 1.15T, 1.20T and 1.25T, where T is the clock period achieved by extreme R&S.

can be improved by up to 4% (average 1.5%) compared with that after extreme R&S. However, the timing performance improvement with the fine-tuning is less than that without the fine-tuning in most cases. One of the reasons is that the commercial tools do not support the timing optimization and analysis of wave-pipelining paths where latches are inserted to block fast signals. Accordingly, only flip-flops are used as delay units to block fast signals in the fine-tuning of VirtualSync+ with Design Compiler. Another reason of this phenomenon is that in the iterative relaxation without the fine-tuning, gates are sized ideally without considering the input transitions and output loads of gates. Therefore, the timing performance improvement without the fine-tuning is a theoretical upper bound of timing performance improvement, which can be used to quickly evaluate the benefit of VirtualSync+ in circuit design. If it is beneficial to apply VirtualSync+, designers can then adopt the proposed fine-tuning method with Design Compiler to generate the optimized circuits. With VirtualSync+, the power consumption and power-delay-area product of the optimized circuits can also be reduced in most cases. The comparison is illustrated in Fig. 13.

In the experiments, the extreme R&S to push the timing performance beyond the limit of traditional sequential designs is realized by iteratively reducing the clock period and optimizing the circuits with retiming and sizing in Design Compiler until the circuits cannot be optimized further anymore. This method squeezes the timing performance at the cost of area overhead. In cases where timing performance does not need to be pushed so far for the sake of area, VirtualSync+ can still outperform the method combining retiming and sizing in terms of area and speed. To demonstrate the effectiveness of VirtualSync+ in such cases, the clock period achieved by extreme R&S, denoted as T, is relaxed, denoted as relaxed R&S. For example, we used the relaxed clock periods $1.05T$, $1.10T$, $1.15T$, $1.20T$ and $1.25T$ to compare VirtualSync+ and relaxed R&S in terms of area and timing performance. With these relaxed clock periods, VirtualSync+ with fine-tuning in commercial tools is applied to improve the timing performance of the circuits optimized with relaxed R&S. The comparison in timing performance and area with respect to the relaxed clock periods is illustrated in Fig. 14, where each pair of symbols, e.g., triangles, represent a comparison in timing and area between VirtualSync+ and relaxed R&S with a specified clock period. For example, in Fig. 14(a), the pair of triangles represent the results with VirtualSync+ and relaxed R&S when the clock period is $1.05T$. Since the result with VirtualSync+ is at the lower left of the result with the relaxed R&S, the timing performance with VirtualSync+ is better than that of relaxed R&S, while the area overhead is reduced. By connecting these symbols together, we can see that in most cases VirtualSync+ achieves a smaller area under the same clock period compared with relaxed R&S. In addition, VirtualSync+ achieves a smaller clock period under the same area cost compared with relaxed R&S.

VIII. Conclusion

In this paper, we have proposed a new timing model, VirtualSync+, in which sequential components and combinational logic gates are considered as delay units. They provide different delay effects on signal propagations on short and long paths. With this new timing model, a timing optimization framework has been proposed to insert delay units only at necessary locations. In addition, we further enhance the optimization with VirtualSync+ by fine-tuning with commercial design tools, e.g., Design Compiler from Synopsys, to achieve more accurate result. Experimental results show that circuit performance can be improved by up to 4% (average 1.5%) compared with that after extreme retiming and sizing, while the increase of area is still negligible. In the future work, we will enhance the framework to integrate VirtualSync+ with physical design tools to fully automate virtual synchronization into the EDA flow. To implement wave-pipelining paths with latches using the commercial tools, however, either the commercial timing tools provide more flexible timing constraints

for these paths or they allow access to the APIs.

REFERENCES

[1] G. L. Zhang, B. Li, M. Hashimoto, and U. Schlichtmann, "VirtualSync: Timing optimization by sychronizing logic waves with sequential and combinational components as delay units," in *Proc. Design Autom. Conf.*, 2018.

[2] Y.-M. Yang, K. H. Tam, and I. H.-R. Jiang, "Criticality-dependency-aware timing characterization and analysis," in *Proc. Design Autom. Conf.*, 2015.

[3] P. Cao, Z. Liu, J. Guo, and J. Wu, "An analytical gate delay model in near/subthreshold domain considering process variation," *IEEE Access*, vol. 7, no. 2019, pp. 171 515 – 171 524, 2019.

[4] T.-Y. Lai, T.-W. Huang, and M. D. F. Wong, "LibAbs: An efficient and accurate timing macro-modeling algorithm for large hierarchical designs," in *Proc. Design Autom. Conf.*, 2017.

[5] H.-H. Cheng, I. H.-R. Jiang, and O. Ou, "Fast and accurate wire timing estimation on tree and non-tree net structures," in *Proc. Design Autom. Conf.*, 2020.

[6] G. L. Zhang, B. Li, and U. Schlichtmann, "PieceTimer: A holistic timing analysis framework considering setup/hold time interdependency using a piecewise model," in *Proc. Int. Conf. Comput.-Aided Des.*, 2016.

[7] N. Chen, B. Li, and U. Schlichtmann, "Iterative timing analysis based on nonlinear and interdependent flipflop modelling," *IET Circuits, Devices & Systems*, vol. 6, no. 5, pp. 330–337, 2012.

[8] C.-P. Chen, C. C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," in *Proc. Design Autom. Conf.*, 1998.

[9] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011.

[10] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *Proc. Int. Conf. Comput.-Aided Des.*, 2012.

[11] H.-R. Jiang, J.-Y. Jou, and Y.-W. Chang, "Noise-constrained performance optimization by simultaneous gate and wiresizing based on lagrangian relaxation," in *Proc. Design Autom. Conf.*, 1999.

[12] C. Lin and H. Zhou, "An efficient retiming algorithm under setup and hold constraints," in *Proc. Design Autom. Conf.*, 2006.

[13] A. P. Hurst, A. Mishchenko, and R. K. Brayton, "Scalable min-register retiming under timing and initializability constraints," in *Proc. Design Autom. Conf.*, 2008.

[14] H.-L. Wang, M. Zhang, and P. A. Beerel, "Retiming of two-phase latch-based resilient circuits," in *Proc. Design Autom. Conf.*, 2017.

[15] D. R. Singh, V. Manohararajah, and S. D. Brown, "Incremental retiming for FPGA physical synthesis," in *Proc. Design Autom. Conf.*, 2005.

[16] H. Cheng, X. Li, Y. Gu, and P. A. Beerel, "Converting flip-flop to clock-gated 3-phase latch-based designs using graph-based retiming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2021, doi: 10.1109/TCAD.2021.3068109.

[17] G. L. Zhang, B. Li, Y. Shi, J. Hu, and U. Schlichtmann, "EffiTest2: Efficient delay test and prediction for post-silicon clock skew configuration under process varaitions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 4, pp. 705–718, 2019.

[18] G. L. Zhang, B. Li, and U. Schlichtmann, "EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers," in *Proc. Design Autom. Conf.*, 2016.

[19] B. Li, N. Chen, and U. Schlichtmann, "Fast statistical timing analysis for circuits with post-silicon tunable clock buffers," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 111–117.

[20] C. Chen, W. Qian, M. Imani, X. Yin, and C. Zhuo, "PAM: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Transactions on Computers*, 2021, doi: 10.1109/TC.2021.3131850.

[21] A. Gupta and D. Siewiorek, "Automated multi-cycle symbolic timing verification of microprocessor-based designs," in *Proc. Design Autom. Conf.*, 1994.

[22] H. Higuchi, "An implication-based method to detect multi-cycle paths in large sequential circuits," in *Proc. Design Autom. Conf.*, 2002.

[23] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-pipelining: A tutorial and research survey," *IEEE Trans. VLSI Syst.*, vol. 6, no. 3, pp. 464–474, Sep. 1998.

[24] D. C. Wong, G. D. Micheli, and M. J. Flynn, "Designing high-performance digital circuits using wave pipelining: algorithms and practical experiences," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 1, pp. 25–46, 1993.

[25] D. A. Joy and M. J. Ciesielski, "Clock period minimization with wave pipelining," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 4, pp. 461–472, 1993.

[26] O. Zografos, A. D. Meester, E. Testa, M. Soeken, P. E. Gaillardon, G. D. Micheli, L. Amarù, P. Raghavan, F. Catthoor, and R. Lauwereins, "Wave pipelining for majority-based beyond-CMOS technologies," in *Proc. Design, Autom., and Test Europe Conf.*, 2017.

[27] J.-C. Shyur, H.-P. Chen, and T.-M. Parng, "On testing wave pipelined circuits," in *Proc. Design Autom. Conf.*, 1994.

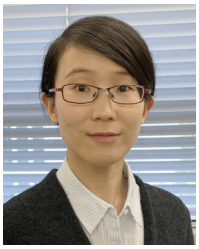
[28] Y. Kra, T. Noy, and A. Teman, "WP 2.0: Signoff-quality implementation and validation of enegery-efficient clock-less wave propagated pipelining," in *Proc. Design, Autom., and Test Europe Conf.*, 2021.

[29] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing," in *Proc. Design, Autom., and Test Europe Conf.*, 2018.

[30] G. L. Zhang, B. Li, M. Li, B. Yu, D. Z. Pan, M. Brunner, G. Sigl, and U. Schlichtmann, "Timingcamouflage+: Netlist security enhancement with unconventional timing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4482–4495, 2020.

[31] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013. [Online]. Available: http://www.gurobi.com

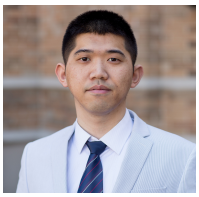
[32] K. Sakallah, T. Mudge, and O. Olukotun, "checkT_c and minT_c: Timing verification and optimal clocking of synchronous digital circuits," in *Proc. Int. Conf. Comput.-Aided Des.*, 1990.



Grace Li Zhang received the Dr.-Ing. degree from the Technical University of Munich (TUM), Munich, Germany, in 2018. She is currently a postdoctoral researcher pursuing Habilitation at the Chair of Electronic Design Automation, TUM, where she leads the research team on heterogeneous computing. Her research interests include neural networks and neuromorphic computing, computer architectures, and machine learning for EDA. She has served/is serving on the technical committee of several conferences including DAC, ICCAD, ASP-DAC, GLSVLSI etc.

Bing Li received the Dr.-Ing. degree from Technical University of Munich (TUM), Munich, Germany, in 2010 and finished the Habilitation there in 2018. He is currently a researcher with the Chair of Electronic Design Automation, TUM. His current research interests include high-performance and low-power design, AI hardware, and emerging systems. He has served on the Technical Program Committees of several conferences including DAC, ICCAD, DATE, ASP-DAC, etc.

Xing Huang received the Ph.D. degree in electronic science and technology from Fuzhou University, Fuzhou, China, in 2018. He is currently a Post-doctoral Research Fellow with the Chair of Electronic Design Automation, Technical University of Munich, Germany, sponsored by the Alexander von Humboldt Foundation. His current research interests include design automation for microfluidic biochips and integrated circuits.



Xunzhao Yin is an assistant professor of the College of Information Science and Electronic Engineering at Zhejiang University. He received his Ph.D. degree from University of Notre Dame in 2019 and B.S. degree from Tsinghua University in 2013, respectively. His research focuses on emerging circuit/architecture designs and novel computing paradigms with both CMOS and emerging technologies.

Cheng Zhuo (M'12–SM'16) received the B.S. and M.S. degrees in electronic engineering from Zhejiang University, Hangzhou, China, in 2005 and 2007, respectively. He received the Ph.D. degree in computer science & engineering from the University of Michigan, Ann Arbor, MI, USA, in 2010. He is currently with Zhejiang University as a Professor in the college of Information Science & Electronic Engineering. His current research interests include computing in memory, deep learning, and general VLSI EDA areas.

Masanori Hashimoto received the Ph.D. degrees from Kyoto University in 2001. He is currently a Professor with the Graduate School of Informatics, Kyoto University. His current research interests include design for reliability, timing and power integrity analysis, reconfigurable computing, soft error characterization, and low-power circuit design.

Ulf Schlichtmann received the Dipl.-Ing. and Dr.-Ing. Degrees in electrical engineering and information technology from Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively. He was with Siemens AG, Munich, and Infineon Technologies AG, Munich, from 1994 to 2003, where he held various technical and management positions in design automation, design libraries, IP reuse, and product development. Since 2003, he is Professor and the Head of the Chair of Electronic Design Automation at TUM. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems. Increasingly, he focuses on emerging technologies such as lab-on-a-chip and photonics.