Online Clock Skew Tuning for Timing Speculation

Rong Ye[†‡], Feng Yuan[†] and Qiang Xu[†‡]
[†]CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[‡]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
Email: {rye,fyuan,qxu}@cse.cuhk.edu.hk

ABSTRACT

The timing performance and yield of integrated circuits can be improved by carefully assigning intentional clock skews to flip-flops. Due to the ever-increasing process, voltage, and temperature variations with technology scaling, however, traditional clock skew optimization solutions that work in a conservative manner to guarantee "always correct" computation cannot perform as well as expected. By allowing infrequent timing errors and recovering from them with minor performance impact, the concept of timing speculation has attracted lots of research attention since it enables "better than worst-case design". In this work, we propose a novel online clock skew tuning technique for circuits equipped with timing speculation capability. By observing the occurrence of timing errors at runtime and tuning clock skews accordingly, the proposed technique is able to achieve much better timing performance when compared to existing clock skew optimization solutions. Experimental results on various benchmark circuits demonstrate the effectiveness of the proposed methodology.

1. INTRODUCTION

Clock skew optimization (CSO) [1] has been exploited as an effective technique to improve the timing performance of integrated circuits (ICs), by assigning intentional clock arrival times to flip-flops (FFs) in synchronized sequential circuits. Earlier works in this domain [1–5] try to find a good clock schedule that maximizes the timing slack of all paths. Recently, with the introduction of tunable clock tree to combat process variation [9], researchers have also presented various post-silicon clock skew tuning techniques to improve circuit timing performance [6, 8, 10–12].

All the above works ensure that circuits can always operate correctly, even in the worst case scenario. With the ever-increasing static process variation effects due to manufacturing imperfection and dynamic variation effects such as voltage and temperature fluctuations, a large guard band needs to be reserved when conducting clock skew optimization, leading to rather limited performance improvement room for such conservative approaches. Instead, timing speculation technique such as Razor [14] allows infrequent occurrence of timing errors and achieves timing error resilience by employing error detection and correction techniques. This "better than worst-case" design methodology enables the tradeoff between reliability and performance/power and hence can achieve much better energy efficiency. It has thus received lots of research attention from both academia and industry [14–16].

To the best of our knowledge, there is no clock skew optimization work for circuits equipped with timing speculation capability. This is unfortunate, because these two techniques naturally complement each other and combining them together is able to achieve much better timing performance. On the one hand, with timing speculation, we do not need to guarantee "always correct" operation during clock skew optimization, which significantly enlarges the improvement room of skew optimization techniques. On the other hand, clock skew tuning can be used to manipulate the occurrence of timing errors in the circuit in such manner that those frequently sensitized paths are with larger timing slack and hence the overall timing error rate of the circuit can be reduced, resulting in better throughput of the circuit.

Motivated by the above, in this paper, we propose a novel online skew tuning framework for circuits equipped with timing speculation capability. To be specific, we develop novel hardware architecture to collect online timing error information and use it to guide our proposed clock skew tuning procedure to achieve better timing performance. Experimental results on various benchmark circuits demonstrate the effectiveness of our proposed methodology. Note that, our framework focus on post-silicon skew tuning, and hence it can easily be combined with pre-silicon skew scheduling works.

The remainder of this paper is organized as follows. In Section 2, we present the preliminaries of this work. The proposed online skew tuning framework and the corresponding algorithms are then detailed in Section 3 and Section 4, respectively. Next, Section 5 presents our experimental results on various benchmark circuits. Finally, Section 6 concludes this paper.

2. PRELIMINARIES AND MOTIVATION

2.1 Pre-Silicon Clock Skew Scheduling

Generally speaking, pre-silicon clock skew scheduling can be classified into two categories by different optimization objectives: performance-driven ones [1, 2] to achieve the highest operational frequency and timing yield-driven ones [3–7, 10] to maximize yield under a particular clock period. Considering the variability of critical path delays, some prior works (e.g., [1, 2]) allocated a safety margin with both upper and lower bounds to each feasible region of clock skews in advance in order to minimize clock period, and some other works (e.g., [4–7]) used statistical models to optimize this problem. All these works try to assign a good clock schedule at design stage, relying on static timing analysis results and process variation models. These offline estimation/analysis techniques, however, cannot provide very accurate timing information and

ICCAD 2011, November 7-10, 2011, San Jose, California, USA.

hence limit the effectiveness of pre-silicon clock skew scheduling solutions.

2.2 Post-Silicon Clock Skew Tuning

Recently, post-silicon tuning capability has been introduced to clock tree design to remove unintentional skews and boost timing yield under increasing process variations [8]. A representative example is Intel's dual-core Itanium processor, which places tunable delay buffers (TDBs) in the clock distribution network to cancel clock skew variations [9]. These TDBs can be programmed from the test access port (TAP).

To realize post-silicon skew tuning, the design of tunable delay buffers is important and various design schemes have been presented in the literature [10]. With the help of TDB designs, the authors of [11] proposed a post-silicon clock timing adjustment strategy, but how to program the tunable elements and determine their locations is not addressed. In [10,12], the authors combined a statistical timing driven skew scheduling algorithm with a post-silicon clock tuning scheme.

The above works rely on offline testing to obtain timing information and use it for post-silicon clock tuning. Path delay test generation, however, is an extremely difficult problem and the coverage is usually quite low. More importantly, with technology scaling, the discrepancy between circuits' timing behavior in functional mode and that in structural test mode has dramatically increased [13]. Due to the above, the effectiveness of existing post-silicon clock skew tuning techniques is also limited.

2.3 Timing Speculation

Circuit-level timing speculation technique, being able to detect timing errors at online stage, react to the error quickly and recover from it by rolling back to a known-good pre-error state, has become one of the most promising solutions to deal with the ever-increasing static and dynamic variation effects with technology scaling. Various techniques have been presented for online timing error detection. Without loss of generality, we employ the representative *Razor* flip-flop (refer to [14] for details) to detect timing errors in this work, by replacing all critical FFs that are driven by speed-paths (i.e., critical or near-critical paths) of the circuit with Razor-FFs.

2.4 Motivation

Targeting circuits equipped with timing speculation capability, this work is motivated by the following observations. A specific manufactured circuit has its unique characteristics (e.g., path delay distribution), which is difficult to estimate during design stage or costly to characterize with delay testing techniques accurately. Consequently, a large design guard band needs to be reserved for conventional clock skew optimization techniques. With timing speculation, however, we do not need to guarantee "always correct" operation, which dramatically increases the flexibility and improvement room of clock skew optimization techniques. While existing timing speculation techniques are able to apply dynamic voltage/frequency scaling to achieve better energy-performance tradeoff using timing error rate information, this is conducted at the entire circuit level. With online clock skew tuning capability, we can manipulate timing error rate in a fine-grained manner so that those frequently sensitized paths are with larger timing slack, and hence reduce the overall timing error rate of the circuit.

The above motivates us to design a novel online clock skew tuning framework, as shown in Fig. 1, wherein we collect runtime timing error information and use it to guide our clock skew tuning process to achieve better circuit performance, as detailed in the following sections.

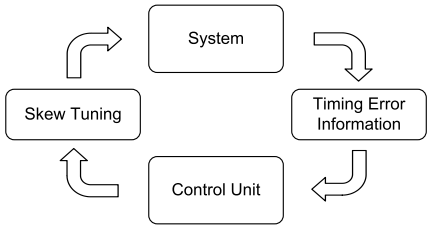

Figure 1: Proposed Online Skew Tuning Framework

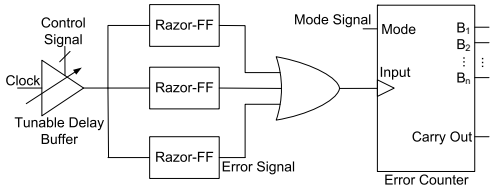

Figure 2: Conceptual Basic Tuning Block

3. DESIGN FOR ONLINE CLOCK SKEW TUNING

Circuit equipped with timing speculation capability can detect and correct infrequent timing errors, but it does not support the collection of timing error information, let alone clock skew tuning. Consequently, we need to add additional circuitries into the design to achieve this objective.

3.1 Basic Tuning Block

To monitor and manage system timing behavior as shown in Fig. 1, the most aggressive design would be to record the timing errors occurred on each Razor-FF and conduct clock skew tuning individually. Clearly, such design will introduce unaffordable hardware cost, and complicate the clock skew tuning procedure. A more practical approach is hence to group adjacent Razor-FFs together to form a basic tuning block, wherein we collect timing errors occurred in it together and we insert only one tunable delay buffer for each block to apply skew tuning. Within each block, all FFs receive the same clock signal, whose skew is controlled by the control unit (see Fig. 1). Note that, due to the above, clock skew tuning will only affect paths between blocks.

Without loss of generality, we assume that the error signal will be set as 1 once timing error is detected by the corresponding Razor-FF. These error signals are "ORed" together and connected to a counter (see Fig. 2). Since the likelihood of having multiple Razor-FFs in a block to have timing errors simultaneously is quite low, the above design saves area cost with little accuracy loss. The carry-out signal of counter can be used to indicate whether the counter is full.

3.2 Timing Error Collection and Clock Skew Tuning Mechanism

One challenging issue in the proposed framework is how to online collect timing error information from all the distributed blocks to the system-level control unit. The most straightforward solution would be to connect the error counter of every block to the control unit. This strategy, however, incurs unaffordable routing cost to the system.

To tackle this problem, we propose a serially shifting mechanism and the proposed architecture is depicted in Fig. 3. In this mechanism, the distributed error counters can be reconfigured to work as a shift register by adding some additional control logic. In other words, the error counter has two operational modes: counting mode and shifting mode. In counting

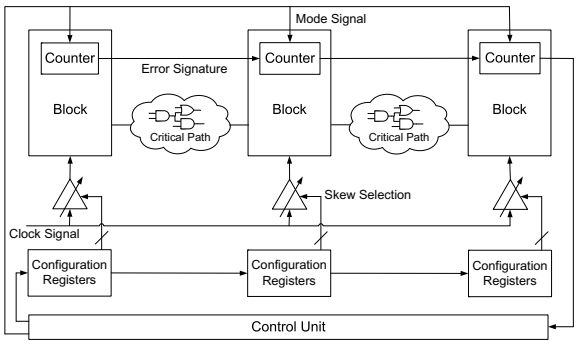

Figure 3: Block Diagram for Timing Error Collection and Clock Skew Tuning

mode, the number of timing errors occurred in the corresponding block is accumulated. Whenever an error counter is full (indicated by the carry out signal), the system-level control unit will receive a "full" signal from this block and then set all the counters into shifting mode to collect timing errors from all blocks. In shifting mode, the FFs in all counters are serially linked as a shift register and their values are shifted out to control unit without disturbing the normal operation of the system. With this mechanism, we are able to collect timing error information in a sampling way during a certain tuning period by assuming that timing error rate in shifting mode is the same with that during counting mode.

At the end of each tuning period T, control unit determines the skew setting for each block based on the collected timing error information, and sends out the corresponding control signals to the tunable delay buffers equipped with each basic tuning blocks, using a similar shifting mechanism. In other words, control signals are serially shifted into the configuration registers and they are applied in parallel when ready for applying skew tuning.

4. PROPOSED ALGORITHMS

4.1 Grouping Algorithm for Tuning Block Formation

Since each tuning block is equipped with only one tunable delay buffer, we would like to have all the FFs within a block to lie in the same subtree[1] from the viewpoint of clock tree structure, so that we can simply insert the TDB in the root node of this subtree. This grouping problem studied here is to determine a grouping plan with maximum system controllability satisfying $n_b/n_f < \beta$, wherein n_b is the total number of blocks after grouping and n_f is total FF count. Here, system controllability is defined as $(1-P_{inside}/P)$, wherein P_{inside} represents the number of critical paths inside blocks and P is the total number of critical paths in the whole circuit. Larger controllability means that more critical paths are outside blocks after grouping. The proposed grouping procedure is a bottom-up approach based on the physical layout of the synthesized clock tree structure, detailed in Fig. 5.

First of all, we initialize all the leaf nodes as subtrees (Line 4∼5), and then try to merge them together to form larger subtrees (Line 6∼10). The decision on whether to merge subtrees or not depends on the critical path count in the merged subtree. To trade off system controllability and hardware cost, we allow a certain number (specified as parameter p) of critical paths inside subtrees. Consequently, the criterion used to

[1]The concept "subtree" used in the context of clock tree means the same concept with "block", which will not be explicitly explained in the following.

#	T_{ij}: the subtree using N_{ij} as root
#	**T**: the set of subtrees
#	**B**: the resulting set of blocks
1.	Initialize $p = 0$
2.	**Repeat**
3.	$p++$
4.	Initialize all leaf nodes as T_{hj}
5.	$\mathbf{T} \leftarrow \{T_{hj} : j = 1, \cdots, m_h\}$
6.	**For** tree level iteration i from $(h-1)$ to 1
7.	**For** node index iteration j from 1 to m_i
8.	**If** all the children subtrees of $T_{ij} \in \mathbf{T}$ **AND** $critical\_path\_number(T_{ij}) < p$
9.	$\mathbf{T} \leftarrow$ Include(T_{ij})
10.	**Else B** $\leftarrow$ Include(all children subtrees of T_{ij})
11.	Exclude all the B-type and C-type blocks from **B**
12.	**Until** $n_b/n_f < \beta$

Figure 5: Proposed Grouping Algorithm

decide merging becomes whether the number of critical paths within a subtree is less than a parameter p (see Line 8): if the critical path count in the new subtree is less than p, we merge its children subtrees together; otherwise, we have to treat its children subtrees as different blocks. By doing so, in the case with $p = 1$, we obtain subtrees without any critical path inside them as shown in Fig. 4 (a), where the subtrees are outlined by the dash line rectangles. We can see that the two wends of any critical path are included in different subtrees.

However, it is not necessary to collect timing errors and add tunable delay buffer for every block. For the ease of discussion, let us first define four types of FFs according to their relationships to critical paths: (i) BR-type FFs, serving as both beginner and receiver of critical path; (ii) R-type FFs, serving as only receiver of critical path; (iii) B-type FFs, serving as only beginner of critical path; (iv) C-type FFs, common FFs, serving as neither beginner nor receiver of critical path. Based on this, we define four types of subtrees/blocks as listed in Table 1 according to the FF types these blocks contain. For example, if a certain block contains no BR-type FF but R-type FF inside, it is defined as R-type block no matter whether it has B-type or C-type FF. Based on this categorization, we have the following observation: for those FFs that are not included in BR-type blocks, their optimal skews can be easily determined offline at design stage and hence there is no need to tune their skews at runtime.

With the above, we can conclude that only BR-type block needs to be equipped with both timing error collection and tuning capabilities, R-type blocks requires to have timing error collection capability only, while the other two types do not need to be observed or tuned at runtime. Therefore, we can get the final set of tuning blocks as shown in Fig. 4 (b) (again, for the case with $p = 1$). By incrementing parameter p, we can obtain grouping outcome with less blocks until satisfying the pre-defined requirement $n_b/n_f < \beta$. To get proper grouping faster, a binary search method can be used to set the parameter p, instead of incrementing its value by one each time.

4.2 Skew Tuning Algorithm

Our proposed clock skew tuning technique is comprised of two phases: (i) offline phase (see Fig. 6) and (ii) online phase (see Fig. 7). By taking online timing error information into consideration, online clock skew tuning can optimize the circuit's timing performance based on the variation characteristics of a specific chip and the actual path sensitization of applications.

Before introducing our algorithms in detail, let us discuss how to setup the skew tuning step for each block first. To guarantee that there exists no silent errors (i.e., timing errors occurs on those FFs that are not protected with Razor-like

Block Type	FF Type				Equipment	
	BR-type FF	R-type FF	B-type FF	C-type FF	Tuning Mechanism	Observing Mechanism
BR-type Block	√	◯	◯	◯	Yes	Yes
R-type Block	×	√	◯	◯	No	Yes
B-type Block	×	×	√	◯	No	No
C-type Block	×	×	×	√	No	No

√: contained; ×: not contained; ◯: do not care.

Table 1: Categorization for Tuning Block Formation

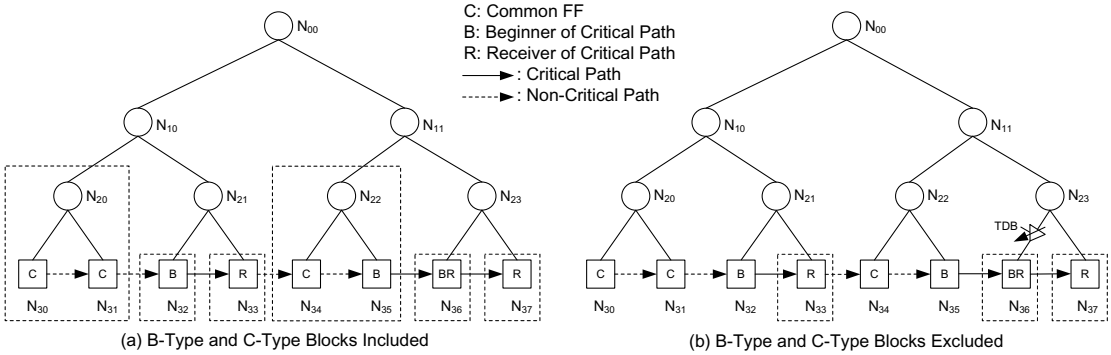

Figure 4: Tuning Block Formation: An Example

#	**C**: the set of BR-type blocks, $\mathbf{C} \subset \mathbf{B}$
#	C_i: BR-type block, $C_i \in \mathbf{C}$
#	**Offline Phase**
1.	**For** the iteration i from 1 to n_f
2.	**If** $FF_i \notin C_j$, for $\forall C_j$
3.	**If** FF_i is B-type FF
4.	Set skew level of $FF_i = 1$
5.	**Else** if FF_i is R-type FF
6.	Set skew level of $FF_i = 5$

Figure 6: Proposed Algorithm in Offline Phase

detectors), we constrain the maximum tunable skew values of TDBs to ensure enough timing margin is kept for non-Razor FFs. If the timing threshold γ (e.g., $\gamma = 80\%$ of clock cycle period) is used to set up Razor-FFs, the maximum tunable skew is

$$S_{max} = [(1-\gamma) \cdot T_c - \delta]/2 \ , \tag{1}$$

wherein T_c is clock cycle period and δ is a safety margin. Five tuning skew levels[2] as shown in Table 2 are used in our implementation, and all the skew levels are set to be 3 initially.

Skew Level	1	2	3	4	5
Skew Value	$-S_{max}$	$-S_{max}/2$	0	$S_{max}/2$	S_{max}

Table 2: Tunable Skew Set

As discussed in Section 4.1, for all those FFs that are not included in BR-type blocks after grouping procedure, their optimal skew tuning can be simply determined at design stage, as shown in Fig 6. Besides, the most important part of proposed tuning algorithm is online phase, a heuristic approach targeting the FFs in BR-type blocks beyond above offline phase. It is, however, rather difficult to determine the skew setting of BR-type blocks, because any action taken to a block is possible to be a double-edged sword: either worsen the block itself to benefit the blocks as the receivers of critical paths starting from this block or improve itself to worsen the receivers. Consequently, we need to determine it at online phase (see Fig. 7). To tackle the above problem, we first define the *benefit* for block k as below,

$$benefit(k) = \sum_{i \in U_k} (error(i) - error(i)^{new}) \ , \tag{2}$$

wherein $error(i)$ is error rate of block i, and U_k is the block set including block k itself and all the blocks as the receivers of critical paths starting from block k. If $benefit(k) \geq 0$, the tuning action to block k in last tuning period is considered to be beneficial, otherwise it is not beneficial.

First of all, we consider those blocks that do not have timing errors during the last tuning period (Lines 1∼15). Since these blocks do not encounter errors, it is very likely that they have extra timing budgets. Our tuning process for these blocks decreases the timing budgets of blocks without errors, while relaxing the timing stress of blocks with errors. After that, our focus is changed to those blocks with errors (Line 16∼27). This procedure is repeated periodically. The proposed online tuning algorithm can be seen as a greedy heuristic, which tries to handle the block with the highest error rate each time. The mechanism to cancel those tuning actions that do not bring any benefits can guarantee to reduce the timing error rates of the overall system step by step.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

To evaluate the effectiveness of the proposed online skew tuning methodology, we conduct experiments on several large ISCAS'89 and IWLS'05 benchmarks. We synthesize these circuits, generate clock tree structures, and obtain timing information using Synopsys EDA tools. To take process variation effect into consideration, we perform Monte Carlo simulation to inject gate-level delay variation following Gaussian distribution with standard deviation equal to 5%.

We set the top 5% longest paths as critical paths and treat their receivers as Razor-FFs, and we assume with the help of error recovery mechanism, we can roll back the system once timing error is detected and we can lower system frequency for a short while to re-compute the result in the failure cycle. Similar with [15], we can trade off timing error rate with clock cycle period to achieve a higher throughput according to the

[2]Tuning forwardly/backwardly means decreasing/increasing the skew level.

```
#   C: the set of BR-type blocks, C ⊂ B
#   C_i: BR-type block, C_i ∈ C
#   Online Phase
1.  Initialize C_i → skew = 3, for ∀C_i
2.  Initialize error(C_i) after 1st tuning period T_1, for ∀C_i
3.  For ∀C_i : error(C_i) = 0
4.      Set C_i → tunable = true
5.      C_i → skew − −
6.  Repeat for each tuning period T_{k_1} (k_1 = 2, 3, ···)
7.      For ∀C_i : error(C_i) = 0
8.          If C_i → tunable = true
9.              If benefit(C_i) ≥ 0
10.                 If C_i → skew ≠ 1
11.                     C_i → skew − −
12.                 Else set C_i → tunable = false
13.             Else C_i → skew + + // cancel last action
14.                 Set C_i → tunable = false
15. Until C_i → tunable = false, for ∀C_i : error(C_i) = 0

16. Set C_i → tunable = true, for ∀C_i : C_i → skew ≠ 5
17. Select C_i with largest error(C_i) AND C_i → tunable = true
18. C_i → skew + +
19. C_old ← C_i
20. Repeat for each tuning period T_{k_2} (k_2 = k_1 + 1, ...)
21.     If benefit(C_old) < 0
22.         C_old → skew − − // cancel last action
23.         C_old → tunable = false
24.     Select C_i with largest error(C_i) AND
        C_i → tunable = true
25.     C_i → skew + +
26.     C_old ← C_i
27. Until C_i → tunable = false, for ∀C_i
```

Figure 7: Proposed Algorithm in Online Phase

following equation,

$$\min_{T_c}[(1 + error(T_c) \cdot penalty) \cdot T_c] \ , \qquad (3)$$

wherein T_c is clock cycle period, $error(T_c)$ is the percentage of cycles to have timing errors, and *penalty* is the penalty due to error cycle occurring. This penalty includes both the cycles of wasted execution that must be discarded when an error occurs and the time spent on checkpointing and re-execution. According to [15], we assume the penalty to be 10 cycles.

We assume there are five discrete tuning levels and we conduct simulation with random inputs in our experiments. The tuning period for the proposed algorithm is set as 100,000 cycles. Note that, longer period can be used in practical applications. The experiments are conducted on a *2.8GHz* PC with *4GB* RAM.

To provide a reasonable baseline reference, we utilize the Useful Skew Technique of IC Compiler to optimize the skews during the CAD process. The minimum clock cycle period reported by IC Compiler considering non-Razor case is denoted as $CSO_{nonrazor}$. The optimal clock cycle period with reasonable error cycle rate, selected according to Equation 3, is denoted as $CSO_{baseline}$ and used as the baseline solution. Note that, any design-phase clock skew scheduling algorithm can be combined with our post-silicon skew tuning framework as an initial solution. The offline phase of proposed skew tuning algorithm is denoted as $CSO_{offline}$ and the online phase is denoted as CSO_{online}.

5.2 Results and Discussion

In Table 3, we present the result for our grouping algorithm. The parameter β used to constrain the total number of blocks (see Section 4.1) is set to be 2% for small scale benchmarks (*s38584* and *s38417*), and 0.5% for large benchmarks (*ethernet* and *des_perf*). Assuming skew tuning algorithm is implemented with software, the proposed architecture (see Fig. 3) is implemented and the additional hardware cost to enable on-line tuning is presented in Column 6 of Table 3. As can be seen, the costs are within 4% for small benchmark circuits and within 1% for larger ones, and they are strongly related to the number of basic tuning blocks in the circuit. The runtime to finish grouping procedure is illustrated in Column 7.

Bench.	TG#	TFF#	RB#	BRB#	Cost(%)	Runtime(s)
s38584	21021	1426	4	15	2.62	0.010
s38417	23949	1636	5	22	3.30	0.026
des_perf	155746	9105	0	39	0.83	2.458
ethernet	164912	10752	17	36	0.76	9.070

TG#: total gate count; TFF#: total FF count;
RB#: R-type block count; BRB#: BR-type block count;
Cost: hardware cost for equipping R-type and BR-type blocks.

Table 3: Experimental Results on Hardware Cost

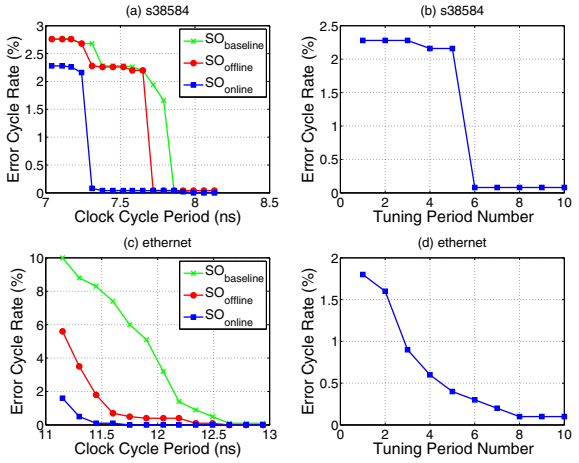

Figure 8: Error Cycle Rates

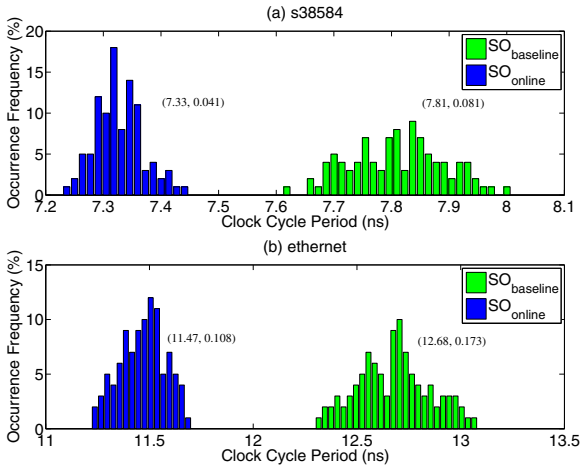

Figure 9: Variation Effects

Firstly, we consider one particular instance of the benchmark circuits (i.e., under a specific variation pattern). In Table 4, CP is the optimal clock period selected according to Equation 3. *Err.* is the corresponding error cycle rate under selected clock period CP, and $Th.$ is the corresponding throughput. We can see that, for *s*38584, the baseline $CSO_{baseline}$ can obtain 4.40% throughput improvement compared with $CSO_{nonrazor}$, a non-Razor solution optimized by IC Compiler. This improvement reflects the efficacy of Razor technique, which is not the main contribution of this work. With error information collection and clock skew tuning capa-

Bench.	$CSO_{nonrazor}$		$CSO_{baseline}$				$CSO_{offline}$				CSO_{online}				
	$CP(ns)$	$Th.(MHz)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_1(\%)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_2(\%)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_3(\%)$	$\Delta_4(\%)$
s38584	8.24	121.36	7.86	0.042	126.69	4.40	7.72	0.047	128.93	1.76	7.31	0.081	135.70	5.25	7.11
s38417	7.93	126.10	7.59	0.097	130.49	3.48	7.47	0.122	132.26	1.36	7.17	0.131	137.67	4.09	5.50
des_perf	15.52	64.43	14.38	0.064	69.10	7.24	14.2	0.047	70.09	1.44	13.25	0.062	75.01	7.01	8.55
ethernet	13.52	73.96	12.64	0.093	78.38	5.98	12.34	0.088	80.33	2.48	11.45	0.106	86.42	7.58	10.25

CP: clock period; $Err.$: error cycle rate under selected clock period; $Th.$: throughput;
Δ_1: throughput difference ratio between $CSO_{baseline}$ and $CSO_{nonrazor}$; Δ_2: throughput difference ratio between $CSO_{offline}$ and $CSO_{baseline}$;
Δ_3: throughput difference ratio between CSO_{online} and $CSO_{offline}$; Δ_4: throughput difference ratio between CSO_{online} and $CSO_{baseline}$.

Table 4: Experimental Results

bilities, offline phase $CSO_{offline}$ has 1.76% improvement compared with $CSO_{baseline}$, and online phase CSO_{online}, again, can achieve extra 5.25% improvement compared with $CSO_{offline}$. Similarly, the results of other benchmark circuits are also listed in Table 4, and we can see that relatively larger improvement can be achieved with larger benchmark circuits.

To have a closer look into the clock tuning process, we present experimental results in Fig. 8, using benchmark circuits *s*38584 and *ethernet* as the representative of small and relatively larger benchmarks, respectively. Fig. 8 (a) and Fig. 8 (c) show that the three curves in each figure have similar trend with respect to increasing cycle period, while the lowest error cycle rate is always achieved by CSO_{online}. Using the skew setting of $CSO_{offline}$ as initial state, CSO_{online} tunes the skews periodically (see Section 4.2 for details). The error cycle rates of first 10 tuning periods under the optimal clock period selected by CSO_{online} are illustrated in Fig. 8 (b) and Fig. 8 (d) to present the change of error cycle rate during online tuning. One notable finding from these figures is that proposed skew tuning can achieve similar decreasing effect of error cycle rate with that of clock period increasing. In other words, by tuning skews we can achieve the error cycle rate as low as the case with laxer clock period.

Finally, we conduct Monte Carlo simulation to produce 100 sample circuits with different variation patterns. Fig. 9 indicates that the mean of selected clock period after applying CSO_{online} is much smaller than that of $CSO_{baseline}$, thanks to clock skew tuning. More importantly, with the same process variation distribution, the clock period distribution of CSO_{online} is with much smaller standard deviation when compared to $CSO_{baseline}$. This is expected, because, unlike offline solutions that can at best optimize the circuit according to a given process variation model, online clock skew tuning takes advantage of the knowledge on each individual chip by timing error collection and facilitates to obtain an optimized chip-specific skew assignment. The corresponding mean clock period and standard deviation of each case are indicated in Fig. 9 in the form of (μ, σ).

6. CONCLUSION

Clock skew optimization is a widely-used technique to improve circuit timing performance, in which we assign intentional clock arrival times to FFs in synchronized sequential circuits. Traditionally, a large timing guard band needs to be reserved due to various variation effects. In this work, with the support of elaborately designed hardware architecture, we propose an online clock skew tuning framework for circuits equipped with timing speculation capability. By observing the occurrence of timing errors at runtime and tuning clock skews accordingly, the proposed technique is able to achieve much better timing performance when compared to existing clock skew optimization solutions, as demonstrated in our experimental results.

7. ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation of China (NSFC) under grant No. 60876029, in part by the General Research Fund CUHK417807 and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

8. REFERENCES

[1] J. P. Fishburn. Clock skew optimization. In *IEEE Transactions on Computers*, vol.39, pp.945-951, July 1990.

[2] R. B. Deokar and S. S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proceeding of International Symposium on Circuits and Systems*, pages 407-410, 1994.

[3] I. S. Kourtev and E. G. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proceeding of International Conference on Computer-Aided Design*, pages 239-243, 1999.

[4] X. Wei, Y. Cai, and X. Hong. Clock skew scheduling under process variation. In *Proc. International Symposium on Quality Electronic Design (ISQED)*, pp. 237-242, 2006.

[5] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for VLSI-chips. In *Proc. the IEEE/ACM International Conference on Computer-aided Design*, pp. 232-238, 1999.

[6] J. L. Tsai, D. H. Baik, C. C. P. Chen, and K. K. Saluja Yield-driven, false-path-aware clock skew scheduling. In *Proc. IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 214-222, 2005.

[7] Y. Wang, W. S. Luk, X. Zeng, J. Tao, C. Yan, J. Tong, W. Cai and J. Ni. Timing yield driven clock skew scheduling considering non-Gaussian distributions of critical path delays. In *Proc. the 45th annual Design Automation Conference*, pp. 223-226, 2008.

[8] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino. Dynamic thermal clock skew compensation using tunable delay buffers. In *Proc. IEEE Transactions on Very Large Scale Integration Sysmtes*, pp. 639-649, June 2008.

[9] P. Mahoney, E. Fetzer, B. Doyle, S. Naffziger. Clock distribution on a dual-core, multi-threaded Itanium?-family processor. In *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 292-293 Vol. 1, 2005.

[10] J. L. Tsai, D. H. Baik, C. C. P. Chen, and K. K. Saluja. A yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *Proc. the IEEE/ACM International Conference on Computer-aided Design*, pp. 611-618, 2004.

[11] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. A post-silicon clock timing adjustment using genetic algorithms. In *Digest of Technical Papers of International Symposium on VLSI Circuits*, pp. 13-16, 2003.

[12] K. Nagaraj and S. Kundu. An Automatic Post Silicon Clock Tuning System for Improving System Performance based on Tester Measurements. In *Proc. IEEE International Test Conference (ITC)*, pp. 1-8, 2008.

[13] S. Sde-Paz and E. Salomon. Frequency and Power Correlation between At-Speed Scan and Functional Tests. In *Proc. IEEE International Test Conference (ITC)*, paper 13.3, 2008.

[14] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceeding of International Symposium on Microarchitecture (Micro)*, pages 7-18, 2003.

[15] M. de Kruijf, S. Nomura, and K. Sankaralingam. A unified model for timing speculation: Evaluating the impact of technology scaling, CMOS design style, and fault recovery mechanism. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pp. 487-496, 2010.

[16] Y. Liu, F. Yuan, and Q. Xu. Re-synthesis for cost-efficient circuit-level timing speculation. In *Proc. Design Automation Conference (DAC)*, 2011.

[17] J. L. Tsai, L. Zhang, and C. Chen. Statistical timing analysis driven post-silicon-tunable clock-tree synthesis. In *Proc. the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pp. 575-581, Nov. 2005.