On Logic Synthesis for Timing Speculation

Yuxi Liu[†], Rong Ye[†], Feng Yuan[†], Rakesh Kumar[§] and Qiang Xu[†]

[†]CUhk REliable Computing Laboratory (CURE)
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {yxliu, rye, fyuan, qxu}@cse.cuhk.edu.hk

[§]Dept. of ECE, University of Illinois at Urbana-Champaign, USA
Email: rakeshk@illinois.edu

ABSTRACT

By allowing the occurrence of infrequent timing errors and correcting them with rollback mechanisms, the so-called timing speculation (TS) technique can significantly improve circuit energy-efficiency and hence has become one of the most promising solutions to mitigate the ever-increasing variation effects in nanometer technologies. As timing error recovery incurs non-trivial performance/energy overhead, it is important to reshape the delay distribution of critical paths in timing-speculated circuits to minimize their timing error rates. Most existing TS optimization techniques achieve this objective with post-synthesis techniques such as gate sizing or body biasing. In this work, we propose to conduct logic synthesis for timing-speculated circuits from the ground up. Being able to manipulate circuit structures during logic optimization, the proposed solution is able to dramatically reduce circuit timing error rates and hence improve its throughput, as demonstrated with experimental results on various benchmark circuits.

1. INTRODUCTION

Technology scaling has brought various challenges to state-of-the-art integrated circuit (IC) design, among which the ever-increasing timing uncertainty caused by static and dynamic variation effects (e.g., manufacturing variability and runtime voltage/temperature fluctuations) is one of the most critical problems [1, 2]. To tolerate such timing uncertainty, conventional designs focus on worst-case parameters and rely on conservative design guardbanding to guarantee "always correct" operations. Such worse-case-oriented design methodology, however, dramatically reduces the benefits brought by technology scaling.

When there is no sufficient timing slack in a circuit, variation effects would manifest themselves as infrequent timing errors on its *speed-paths* [6, 7]. If we could detect the occurrence of timing errors and correct them on-the-fly with little penalties, we can achieve error-resilient computing with improved circuit performance and/or energy-efficiency. Such "better-than-worst-case" design methodology has attracted lots of research attention from both academia and industry, and a number of so-called *timing speculation* (TS) techniques have been presented in the literature [3, 4, 5].

Since timing error detection and correction mechanisms are the enabling techniques in timing-speculated circuit designs, there have been numerous research works to address these two issues. On one hand, many timing error detectors were presented in the literature (e.g., [8, 3, 9]), among which *Razor flip-flop* [3] with double-sampling capability is the most representative one. On the other hand, timing error recovery is usually achieved by restoring system to a known-good pre-error state with microarchitectural-level support. For example, when a timing error is detected in microprocessor datapath, the processor pipeline can be flushed. Then, by lowering the system frequency for a short period and replaying instructions, the processor is able to recover from the failure cycle and continues its operations correctly with performance/power penalties paid for recovery and re-execution. Since the system operates at higher frequency without timing errors for the general case, the overall system performance is expected to be improved, as long as timing errors do not occur frequently and the penalties paid to correct them are well controlled.

Since timing error correction incurs non-trivial performance overhead, it is important to reduce the timing error probability of the circuit under a certain operational frequency to improve its throughput. Various techniques (e.g., [11, 12, 13, 14]) have been presented in the literature to achieve this objective, but most of them are conducted using post-synthesis optimization techniques such as gate sizing or body biasing. The effectiveness of these solutions is thus limited with fixed circuit structure at this stage. Motivated by the above, we propose to conduct logic synthesis for timing speculation from the ground up in this work. With the flexibility to manipulate circuit structural changes, our solution can dramatically improve the throughput of timing-speculated circuits, as demonstrated on various benchmark circuits. The main contributions of this work include:

- We propose a simple yet effective model to estimate the impact of structural changes on the timing error probability of a circuit, by taking process variation effects and speed-path sensitization probability into consideration;
- Using the proposed optimization metrics, we present novel logic synthesis techniques to improve the performance of timing-speculated circuits.

The remainder of this paper is organized as follows. Section 2 presents the preliminaries of this work. In Section 3, we detail the proposed logic synthesis techniques for timing speculation. Experimental results on various benchmark circuits are then presented in Section 4. Finally, Section 5 concludes this paper.

2. PRELIMINARIES

2.1 Timing Speculation

By detecting timing errors on-the-fly and conducting rollback error recovery, the circuit's throughput and/or energy consumption can be significantly improved with timing speculation technique. One of the most representative solutions is the so-called Razor technique [3], wherein double-sampling sequential elements are used for timing error detection while counterflow pipelining technique [10] is used for error recovery.

Assuming the clock period is T and the circuit timing error rate with respect to T is $P_e(T)$, we can have the probability function for the circuit to operate without timing errors $P(T) = 1 - P_e(T)$. By using the timing speculation penalty factor r to indicate that r clock cycles are needed to recover the system and re-execute the failure cycle, the overall system throughput is [11]:

$$TP(T) = \frac{1}{T} \times (1 - P_e(T) + \frac{P_e(T)}{r}) \qquad (1)$$

In traditional "worst-case-oriented" designs, no timing error is allowed and hence we have $P_e(T_{wc}) = 0$, where T_{wc} is the operational clock period in the worst case to guarantee error-free computation. Then, Eq. 1 is simplified to be $TP(T_{wc}) = 1/T_{wc}$, the familiar case with relatively low throughput, because T_{wc} has to be large enough to tolerate variation effects.

With timing speculation, the clock period T can be reduced from T_{wc} to T_{ts} and we have $P_e(T_{ts}) > 0$. From Eq. 1, it is clear that the timing error rate has to be kept within a small range to achieve performance benefit. Generally speaking, the penalty factor r is a fixed value, and we can find an optimal clock period with maximum throughput for a given design. On the other hand, under a certain operational clock period, as can be seen from Eq. 1, the circuit throughput is determined by $P_e(T)$. Consequently, if we can make design changes to reduce circuit timing error rate, the system throughput can be enhanced.

To achieve the above objective, DynaTune [11] optimizes frequently-sensitized critical paths of the circuit by assigning low threshold voltage V_t to some critical gates on them. Blueshift [12] identifies and optimizes frequently-exercised critical paths by On-demand Selective Biasing (OSB) and Path Constraint Tuning (PCT). In [13], *Kahng et al.* proposed a slack re-distribution strategy to increase the level of over-scaling under a given timing error probability constraint to minimize power consumption. They increase the timing slacks of frequently-sensitized critical paths by sizing the on-path gates.

The effectiveness of the above works relies on the circuit generated from the logic synthesis step. To optimize timing-speculated circuits with more flexibility, *Cong et al.* proposed to conduct logic synthesis for timing speculation in [15]. However, in this work, the authors simply changed the cost function used in logic optimization, by taking timing error probability into consideration. In addition, timing simulation is required in each optimization step to acquire switching probability and during synthesis the circuit structure is changed frequently. As a result, its computational complexity is extremely high and it would be too time-comsuming.

2.2 AIG-Based Logic Synthesis

Logic synthesis is a process by which an abstract form of desired circuit behavior is turned into an optimized logic gate implementation (e.g., in terms of timing and/or area), which plays an important role in today's IC design flow.

During logic synthesis, we need to make frequent local structural changes for circuit optimization. Consequently, an effective representation of combinational logic that enables fast local transformation is crucial. And-Inverter Graphs (AIGs), a networks of two-input ANDs and inverters, is one of the most popular such representations [16]. With this representation, structural changes are conducted at each super-gate during the optimization process [17]. By applying the AIG rewriting many times, the structural change scope will be no longer local and it is stated that the cumulative effect for multiple rounds of AIG rewriting is usually superior to traditional synthesis in terms of quality.

DEFINITION 1. ***Super-gate*** *rooted at a* ***node o*** *is composed of a set of nodes in* ***node o****'s subtree whose leaves are either inverters or PIs.*

As shown in Fig. 1(a), the subtree within the dotted line is a super-gate rooted at *Node o*. For this example circuit, let us use AT_a, AT_b, AT_c and AT_d to represent the arrival times of fan-in a, b, c, and d, respectively. Assume $AT_a > AT_b > AT_c > AT_d$, if we change the circuit structure so that fan-in a, the latest-arrived signal, is located to be the closest to the output (see Fig. 1(b)), the circuit delay can be reduced. In other words, by reordering the fan-ins of super-gates, the circuit shown in Fig. 1(b) has better timing performance when compared to that shown in Fig. 1(a) while maintaining the functionality of the circuit.

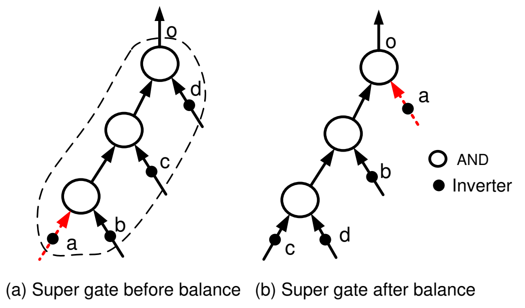


Figure 1: Path delay balancing: an example.

Conventional logic synthesis techniques (e.g., [18, 19]) use the above method to balance the timing of different paths to minimize the worst-case timing delay of the circuit. In timing-speculated circuits, we can apply similar technique to reshape circuit path delay distribution to achieve better performance, but our objective is to improve the overall system throughput instead of the worst-case delay. To achieve this objective, however, is a challenging task, and we detail our proposed solution in the following section.

3. LOGIC SYNTHESIS FOR TIMING SPECULATION

According to Eq. 1, the system throughput of a timing-speculated circuit is a function of its clock period and timing error rate. During logic synthesis, we can make frequent structural changes for circuit optimization. Such flexibility provides us better opportunity to reduce circuit timing error rate under a certain clock period when compared to those post-synthesis optimization solutions. At the same time, however, this objective can be achieved only when we are able to efficiently and effectively evaluate the impact of structural changes on circuit timing error rate. Consequently, we need an optimization metric that can be quickly calculated and use it to guide our logic synthesis procedures.

3.1 Proposed Optimization Metric

There are three factors that affect the overall timing error rate of a circuit: (i). the clock period; (ii). the delay distribution of critical paths; and (iii) the sensitization probability of critical paths. At the logic synthesis stage, it is impossible to obtain the first two factors accurately because they are significantly affected by later stages of the design flow (e.g., technology mapping and physical design). Fortunately, as we just make local changes (e.g., for one super-gate a time) in each logic optimization step, we only need to estimate the corresponding local timing error rate changes. In other words, it is not necessary to acquire an accurate overall circuit timing error rate. Instead, we only need to evaluate the impact of local structural changes on timing error rates. To be specific, when conducting fan-in reordering for a particular super-gate during logic synthesis, the number of critical paths that go through this gate and/or their timing delay will change, and we need an optimization metric that can reflect the above factors.

Timing errors manifest themselves on critical paths whose delays may exceed clock period. Considering process variation, the delay of each gate is a random variable. By assuming the gate delay follows Gaussian Distribution,the delay distribution of a path also follows Gaussian Distribution, which can be represented as (μ, σ), where μ is the mean value of the path delay and σ is the standard deviation of the path delay. Therefore, the probability for the delay of path i to exceed clock period T is:

$$D_i = 1 - \Phi(\frac{T-\mu_i}{\sigma_i}) \quad . \tag{2}$$

The other condition for timing error to occur on a critical path is that this path is sensitized. Previous works (e.g., [15]) resort to timing simulation to acquire path sensitization probability. While being more accurate, it is too time-consuming to be used in the iterative logic optimization procedure. Consequently, we use a simplified model to calculate path sensitization probability. Generally speaking, sensitizing a path requires that there is a signal transition at the input of the path and all the side-inputs of those gates along the path have non-controlling value (e.g., logic '1' for AND gates) at the same time. Eq. 3 presents the sensitization probability of a path i, where NCV represents "non-controlling value", Tog_i and $P(g_j = NCV)$ are the input signal toggle probability and the probability for side-input j along path i to be non-controlling value. Note that, even though the side-inputs along a certain path may not be mutually independent, we ignore this effect and estimate the sensitization probability as shown in the equation to reduce computational complexity.

$$\begin{aligned} S_i &= Tog_i \times P(g_1 = NCV, g_2 = NCV, ..., g_k = NCV) \\ &\approx Tog_i \times P(g_1 = NCV) \times P(g_2 = NCV) \times ... \times P(g_k = NCV) \end{aligned} \tag{3}$$

We obtain the toggle probabilities of the path inputs (FF or PI) by performing one-time logic simulation. This is possible because logic synthesis only changes the structure of combinational logic and hence the toggle probabilities of path inputs would not change during each optimization step. However, we cannot afford to use simulation to obtain $P(g_j = NCV)$ in the above equation due to logic structural changes in each optimization step. Consequently, we acquire this information by performing simple probability propagation. To be specific, we travel through the circuit network in a topological order from PIs to POs and calculate the output's logic probability for one gate according to its inputs' logic probabilities. For example, if the probabilities for the two inputs of a 2-input AND gate to be logic 1 are 0.4 and 0.5 respectively, the probability for its output to be logic 1 should be $0.4 \times 0.5 = 0.2$ and the probability to be 0 is $1 - 0.2 = 0.8$. Note that, we initially assign 0.5 as the probabilities for all PIs to be logic 0 or 1 respectively.

Based on the above, we can estimate the probability that there is timing error to occur on a path i as follows:

$$P_i = D_i \times S_i \quad . \tag{4}$$

When conducting fan-in reordering on a super-gate, there may be multiple critical paths that go through the very same gate. We use the following term as our optimization metric during logic synthesis:

DEFINITION 2. ***Accumulated timing error probability (ATEP)*** *of a gate equals the sum of the timing error probabilities of all critical paths going through it.*

3.2 Proposed Logic Synthesis Solution

With the above optimization metric that can reflect the impact of fan-in reordering of super-gates, there are two key problems that need to be investigated in our optimization procedure: (i) we need to determine how to optimize each super-gate by reordering its fan-ins so that the ATEP of this super-gate can be optimized locally; (ii) we need to investigate what order we should follow to optimize the super-gates one by one.

3.2.1 Fan-In Reordering

To reorder the fan-ins of a certain super-gate, we always select the fan-in permutation that can achieve the lowest ATEP after optimization. Generally speaking, we reorder the fan-ins by trying all possible fan-in permutations to find the best order. However, if the number of fan-ins is large, this enumeration process to investigate all permutations will lead to a large runtime to. Consequently, for those super-gates with a relatively large number of fan-ins, we resort to a heuristic-based grouping method to get a near-optimal order. With the best fan-in order, we calculate the change of the super-gate's ATEP and consider it as an approximation to $benefit$, which is used to demonstrate the maximum potential improvement by reordering the fan-ins of a super-gate. The $benefit$ is defined as follows:

DEFINITION 3. *For a super-gate, different fan-in order will lead to different ATEP. The ATEP difference between the lowest ATEP with the best order and the original ATEP before reordering is defined as this super-gate's* ***benefit****.*

The heuristic-based grouping method is like this. Given a super-gate with n fan-ins, we calculate the ATEPs of all the fan-ins and firstly reorder all of them according to the rule that the fan-in with higher ATEP should be located closer to the super-gate's output. After that, we divide all the reordered fan-ins into k groups evenly with each group containing (n/k) fan-ins. Then we search all possible permutations of the (n/k) fan-ins within each group. By doing so, we find out an n-fan-in permutation with the lowest ATEP. By doing this, we only enumerate the fan-in permutations within each group and the fan-in number in each group is small. The obtained permutation near-optimal and we use it to calculate the approximate $benefit$.

3.2.2 Super-Gate Ordering

During the optimization process, the structural change happens on each super-gate. Reordering the super-gate's fan-ins may provide benefit for the timing error probability. This fan-in reordering process is conducted for each super-gate one by one. As a result, it is necessary to study the impact of the super-gate's order on the result.

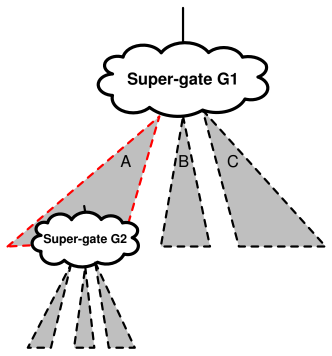


Figure 2: The impact of optimization order of super-gates.

Let us examine a simple example to show how the optimization order of super-gates can affect the final results. As shown in Fig. 2, we have two super-gates (G_1 and G_2) and G_2 is in the fan-in cone A of G_1. Reordering either one of them can reduce the error probability. However, a different optimization order may result in different effectiveness. Assume the ATEP of fan-in A is larger than that of the other two fan-ins B and C. If G_1 is proposed to be optimized first, it is obvious that we should place fan-in A to the position closest to G_1's output to reduce its delay and error probability. However, if G_2 is proposed to be optimized first, it is possible that after the optimization on G_2 the ATEP of fan-in A becomes smaller than the other two fan-ins B and C. Hence it is not necessary to locate fan-in A to the position closest to the output when optimizing G_1. From this example, it can be seen that the optimization result may be different if we follow a different super-gate's order. How to find out an appropriate super-gate's order is an important problem and will be investigated in the following.

First of all, it is intuitive that those super-gates with larger *benefit* should be optimized first. It is because larger *benefit* means these super-gates are expected to achieve more error probability reduction. Dur-

```
#    {G_i}, the set of the super-gates within the subtree.
1.   Initialize the set $\{G_i\}$
2.   Update the benefits of all the super-gates in $\{G_i\}$
3.   Update the flexibilities of all the super-gates in $\{G_i\}$
4.   REPEAT
5.       IF $\{G_i\} = \varnothing$
6.           Break;
7.       ELSE
8.           Select the super-gate $G \in \{G_i\}$ with the smallest flexibility;
9.           Reorder the fan-ins of the super-gate $G$;
10.          Remove the super-gate $G$ from $\{G_i\}$;
11.  END REPEAT
```

Figure 3: The algorithm to optimize the subtree with a super-gate as root.

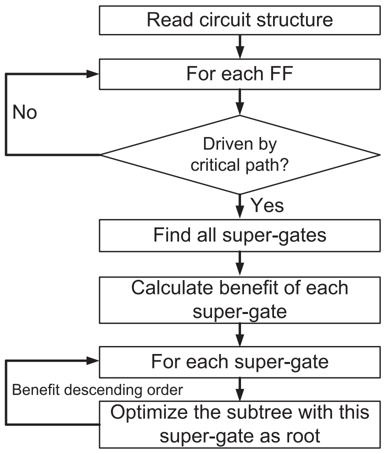


Figure 4: The overall optimization flow.

ing the optimization process, we set up a *benefit* threshold η and optimize the super-gates one by one in a *benefit*-descending order. Once the $benefit$ of currently optimized super-gate is less than the pre-defined threshold η, we would stop the optimization. By doing so, we can avoid the optimization to some super-gates that have too less impact on the error probability reduction and hence save runtime cost.

Secondly, once a super-gate with large enough $benefit$ is selected to be optimized, simply reordering its fan-ins at once may result in a sub-optimal solution as discussed in the above example shown in Fig. 1. Consequently, we propose a novel algorithm to optimize the super-gates, which can effectively avoid the sub-optimal case. Specifically, we define a metric $flexibility$ to indicate the priority of super-gates during optimization process as follows:

DEFINITION 4. *Given a suspicious FF, we can find out all the super-gates in its fan-in cone and construct a tree with it as root and all the super-gates as nodes. The **flexibility** of a super-gate G is defined as the sum of the benefits of all the super-gates within the subtree rooted at the super-gate G.*

The *flexibility* of a super-gate indicates how much its ATEP can be affected by other super-gates. In other words, the larger a super-gate's *flexibility* is, the more probable it is that the super-gate's ATEP will be affected by other super-gates. Consequently, when a super-gate G with large enough $benefit$ is selected for optimization, we would prefer to optimize all super-gates within the subtree S_G[1] first, so that we can get a better understanding to the ATEP of G and finally achieve an effective fan-in reordering for G. Based on the above, we propose an algorithm to optimize the subtree when a super-gate G is selected to be optimized as described in Fig. 3. Before the the subtree optimization, we need to update the *benefits* and *flexibilities* first. Then we start to optimize the super-gates in the subtree in *flexibility*-ascending order. This optimization order can guarantee a bottom-up traversal to the subtree with super-gate G as root, which can avoid the sub-optimal case in the example of Fig. 1. Note that, in order to avoid the optimization to some super-gates once again, if some super-gates have been optimized and the $benefits$ have also been updated already, we will mark out these super-gates and not conduct optimization for them again.

[1]A subtree S_G implies that this subtree is constructed with super-gate G as root.

3.2.3 Overall Optimization Flow

To sum up, the overall optimization flow of our proposed logic synthesis technique is shown in Fig. 4. Firstly, we read the circuit netlist structure into our optimization program and perform timing analysis to differentiate the suspicious FFs driven by critical paths. Secondly, for each suspicious FF, we explore its fan-in cone and find out all the super-gates within it and their *benefits*. Finally, we optimize each super-gate together with the super-gates within its subtrees one by one with $benefit$-descending order.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

We develop our logic synthesis tool for timing speculation based on ABC [17]. To evaluate its effectiveness, we conduct experiments on ISCAS'89 benchmark circuits with UMC's 130nm technology, and perform Monte Carlo simulation to inject gate-level delay variation following Gaussian distribution with standard deviation equal to 10%. To get timing error probability, we perform simulation with random inputs in our experiments and each simulation is performed with 100,000 cycles. Note that, our simulation is conducted on post-layout netlist to incorporate the impact of technology mapping and physical design on timing error rates. The penalty factor r in Eq. 1 is assumed to be 10 clock cycles according to [23]. The baseline solution optimized by ABC, aiming at timing balance, is denoted as $LS_{baseline}$. Our proposed logic synthesis technique, aiming at reducing timing error probability, is denoted as $LS_{proposed}$. We sweep the operating clock period for each case to find out the best one with largest throughput calculated according to Eq. 1.

4.2 Results and Discussion

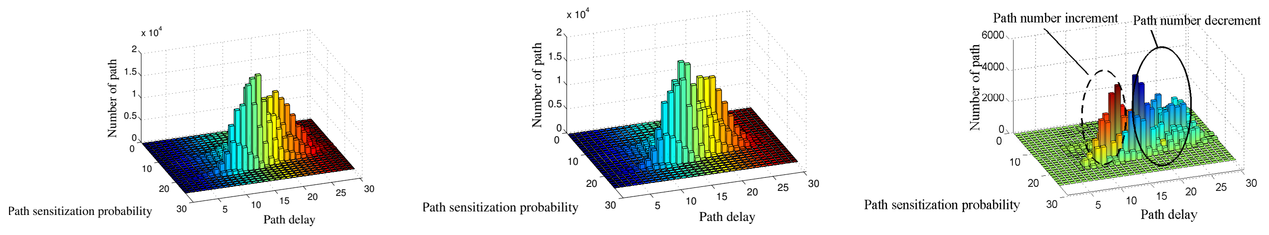
We first present the results on system throughput and hardware cost in Table 1. The throughputs of $LS_{baseline}$ and $LS_{proposed}$ are reported in Column 2 and Column 3, respectively. The throughput improvement between $LS_{baseline}$ and $LS_{proposed}$ is shown in Column 4. We can observe about 17.51% throughput improvement on average after our optimization. Column 7 and Column 8 demonstrate the circuit area of $LS_{baseline}$ and $LS_{proposed}$. The additional hardware cost of $LS_{proposed}$ is shown in Column 9. Only 1.13% hardware cost on average is needed to realize our proposed optimization techniques, which is quite small especially considering the significant improvement of system throughput.

In Column 5 we show the throughput results with our proposed techniques and a special technology mapping step $LS_{baseline} + TM$. This mapping step is different from the original mapping technique in ABC which is used in our proposed logic synthesis technique $LS_{proposed}$ to get the results in Column 3. The optimization target is changed from the area/power to timing error probability to reduce the error rate. For the traditional technology mapping implemented in ABC, the target is to minimize the worst case path delay or reduce the overall circuit area. However, here we make some modifications on the target function. Our target is no longer the worst case delay or overall area. Instead, we try to reduce the timing error rate. As a result, while selecting the coverings in the algorithm, we change the strategy to select the one that can shorten those critical paths which have large contribution on timing errors. The objective to combine our proposed logic synthesis techniques with technology mapping is to indicate that our proposed logic synthesis techniques can be well combined with other optimization techniques to further improve the performance of the final circuit.

Table 1: Results on system throughput and hardware cost.

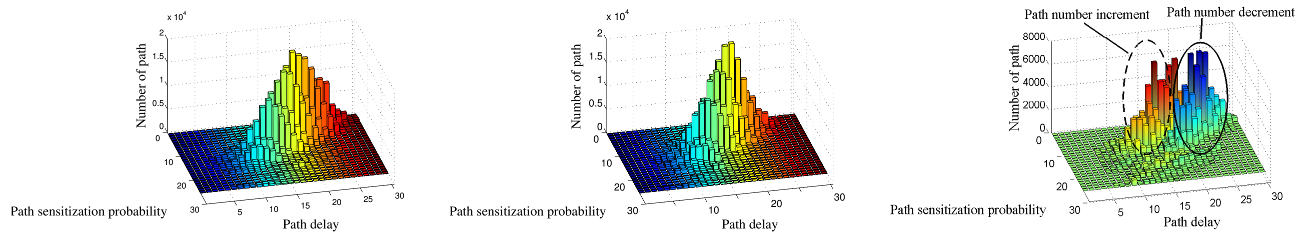
Circuit	Throughput(MHz)					Hardware		
	$LS_{baseline}$	$LS_{proposed}$	$\Delta_1(\%)$	$LS_{proposed}+TM$	$\Delta_2(\%)$	$LS_{baseline}$	$LS_{proposed}$	Cost(%)
s298	456.63	508.36	11.33	540.10	18.27	1206	1210	0.33
s344	283.24	338.64	19.56	349.83	23.51	1823	1840	0.93
s349	285.72	344.27	20.49	347.79	21.72	1878	1886	0.43
s382	556.11	657.78	18.28	672.68	20.96	2569	2578	0.35
s444	422.77	455.25	7.68	466.30	10.30	3089	3121	1.04
s526	456.86	510.42	11.72	530.42	16.10	3327	3363	1.08
s641	210.99	271.27	28.57	276.69	31.14	5671	5779	1.90
s713	243.14	260.23	7.03	272.71	12.16	5798	5852	0.93
s953	343.47	394.52	14.86	401.41	16.87	5841	5915	1.27
s1196	263.42	305.60	16.01	317.29	20.45	6760	6895	1.99
s1238	254.09	325.12	27.95	331.78	30.58	7068	7164	1.36
s1423	271.54	338.65	24.71	345.33	27.17	7444	7488	0.59
s13207	234.39	280.84	19.82	289.4	23.47	30320	30815	1.63
s35932	269.90	303.97	12.62	317.11	17.49	118790	120215	1.20
s38584	162.93	198.77	22.00	203.76	25.06	97901	99706	1.84
AVERAGE			17.51		21.02			1.13

Δ_1: Throughput improvement ratio between $LS_{baseline}$ and $LS_{proposed}$;
Δ_2: Throughput improvement ratio between $LS_{baseline}$ and $LS_{proposed}+TM$.



(a) $LS_{baseline}$ (b) $LS_{proposed}$ (c) $Comparison$

Figure 6: Path delay distribution and sensitization probability on $s35932$.



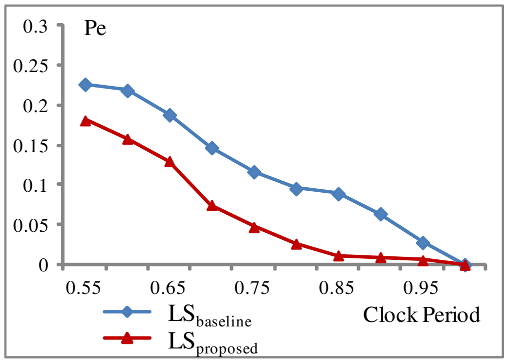
(a) $LS_{baseline}$ (b) $LS_{proposed}$ (c) $Comparison$

Figure 7: Path delay distribution and sensitization probability on $s38584$.

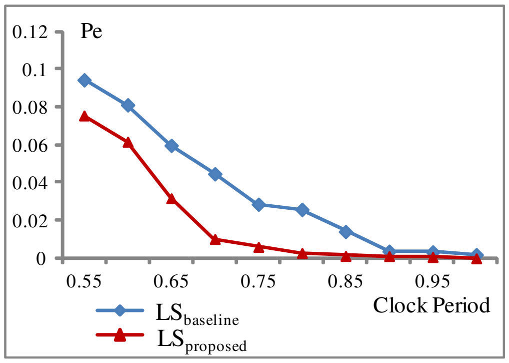
Column 6 shows the improvement compared with $LS_{baseline}$. From the results in Column 4 and Column 6, it can be observed that we can get additional 3.51% improvement on average if we conduct explicit optimization for timing speculation in post-synthesis stages, which demonstrate that our proposed solution can be combined with other optimization techniques for timing speculation (e.g., [13, 14, 24, 25]).

To further examine the effectiveness provided by our proposed solution, we take $s35932$ and $s38584$ as examples to show the change of the delay distribution and the sensitization probability after applying our proposed techniques. In Fig. 6 and Fig. 7, the x-axis describes the path delay distribution[2], the y-axis indicates the path sensitization probability and the z-axis represents the number of paths. Fig. 6(a)(Fig. 7(a)) and Fig. 6(b)(Fig. 7(b)) show the path distribution of $LS_{baseline}$ and $LS_{proposed}$, respectively. To make the difference clear, we compare the path distribution of $LS_{baseline}$ and $LS_{proposed}$, obtain the path number difference between them and present them in Fig. 6(c) and Fig. 7(c), wherein the red bars within the dashed circle represent the increment of path number and the blue ones within the solid circle represent the decrement of path number after applying our optimization technique.

[2]For simplicity, we only present the top 50% longest paths here.



(a) Timing error probabilities on *s*35932.



(b) Timing error probabilities on *s*38584.

Figure 5: Comparison between $LS_{baseline}$ and $LS_{proposed}$.

From these two figures, we can see that the number of the paths with smaller delay or sensitization probability is increased while the number of the paths with larger delay or sensitization probability is decreased, which demonstrate why the proposed technique is effective.

Finally, we show the changes of timing error probabilities with respect to clock period (set as a percentage of the longest path delay) in Fig. 5. Again, from this figure, we can observe that for many of the different clock periods, the timing error probability for $LS_{proposed}$ is reduced when compared with that of $LS_{baseline}$, which is the reason that the circuit throughput is improved. This also proves the effectiveness of the proposed solution.

5. CONCLUSION

In this paper, we propose to conduct logic synthesis for timing speculation from the ground up. Being able to change the logic structure to reduce timing error probability, the proposed solution facilitates to improve the throughput and/or energy-efficiency of circuits equipped with timing speculation capability.

6. ACKNOWLEDGEMENTS

This work was supported in part by the Hong Kong SAR Research Grants Council under General Research Fund No. CUHK418111.

7. REFERENCES

[1] S. Borkar, et al., “Parameter variations and impact on circuits and microarchitecture,” in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2003, pp. 338–342.

[2] K. Bowman, et al., “Circuit techniques for dynamic variation tolerance,” in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2009, pp. 4–7.

[3] D. Ernst, et al., “Razor: a low-power pipeline based on circuit-level timing speculation,” in *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 7–18.

[4] B. Greskamp and J. Torrellas, “Paceline: Improving single-thread performance in nanoscale cmps through core overclocking,” in *Proc. International Conference on Parallel Architecture and Compilation Techniques*, 2007, pp. 213–224.

[5] L. Benini, E. Macii, M. Poncino, and G. D. Micheli, “Telescopic units: A new paradigm for performance optimization of vlsi designs,” *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 220–232, 1998.

[6] D. Frank, R. Puri, and D. Toma, “Design and CAD Challenges in 45nm CMOS and beyond,” in *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 329–333.

[7] S. Borkar, “Designing reliable systems from unreliable components: the challenges of transistor variability and degradation,” *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[8] C. Metra, M. Favalli, and B. Ricco, “On-line detection of logic errors due to crosstalk, delay, and transient faults,” in *Proc. IEEE International Test Conference (ITC)*, 1998, pp. 524–533.

[9] M. R. Choudhury and K. Mohanram, “TIMBER: Time borrowing and error relaying for online timing error resilience,” in *Proc. Design, Automation, and Test in Europe (DATE)*, 2010, pp. 1554–1559.

[10] R. Sproull, I. Sutherland, and C. Molnar, “The counterflow pipeline processor architecture,” *IEEE Design & Test of Computers*, vol. 11, no. 3, p. 48, 1994.

[11] L. Wan and D. Chen, “Dynatune: circuit-level optimization for timing speculation considering dynamic path behavior,” in *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2009, pp. 172–179.

[12] B. Greskamp, et al., “Blueshift: Designing processors for timing speculation from the ground up,” in *IEEE International Symposium on High Performance Computer Architecture*, 2009, pp. 213–224.

[13] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, “Slack redistribution for graceful degradation under voltage overscaling,” in *Proc. Asia and South Pacific Design Automation Conference*, 2010, pp. 825–831.

[14] Y. Liu, F. Yuan and Q. Xu, “Re-synthesis for cost-efficient circuit-level timing speculation,” in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2011, pp. 158–163.

[15] J. Cong and K. Minkovich, “Logic synthesis for better than worst-case designs,” in *Proc. International Symposium on VLSI Design, Automation and Test*, 2009, pp. 166 –169.

[16] A. Mishchenko, S. Chatterjee, and R. Brayton, “DAG-aware AIG rewriting a fresh look at combinational logic synthesis,” in *Proc. Design Automation Conference*, 2006, pp. 532–535.

[17] ABC: http://www.eecs.berkeley.edu/ alanmi/abc/.

[18] J. Cortadella, “Timing-driven logic bi-decomposition,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 675 – 685, June 2003.

[19] A. C. Ling, J. Zhu, and S. D. Brown, “Delay driven AIG restructuring using slack budget management,” in *Proc. ACM Great Lakes symposium on VLSI*, 2008, pp. 163–166.

[20] K.C. Chen, et al., “DAG-map: graph-based FPGA technology mapping for delay optimization,” *IEEE Design & Test of Computers*, vol. 9, no. 3, pp. 7 –20, Sep. 1992.

[21] K.J. Singh, “Timing optimization of combinational logic,” in *Proc. International Conference on Computer-Aided Design (ICCAD)*, 1988, pp. 282–285.

[22] S. Chatterjee, et al., “Reducing structural bias in technology mapping,” in *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 519–526.

[23] M. Kruijf, S. Nomura, 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*, 2010 , pp.487-496.

[24] R. Ye, F. Yuan and Q. Xu, “Online clock skew tuning for timing speculation,” in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 442–447.

[25] R. Ye, F. Yuan, H. Zhou and Q. Xu, “Clock skew scheduling for timing speculation,” in *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, 2012, pp. 929–934.