Optimization for Timing-Speculated Circuits by Redundancy Addition and Removal

Yuxi Liu, 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
Email: {yxliu, rye, fyuan, qxu}@cse.cuhk.edu.hk

***Abstract*—Integrated circuits suffer from severe variation effects with technology scaling, making their timing behavior increasingly unpredictable. Timing speculation is a promising technique to tackle this problem with the help of online timing error detection and correction mechanisms. In this paper, we propose to use redundancy addition and removal (RAR) technique to optimize timing-speculated circuits. By intentionally removing wires on those frequently-exercised critical paths and replacing them with wires on less critical ones (if possible), the proposed technique is able to greatly reduce the timing error rate of the circuit and improve its overall throughput, as shown in our experimental results on various benchmark circuits.**

I. Introduction

The increasing uncertainty of circuit timing behavior resulting from variations has become one of the most serious problems with the continuous shrunk transistor feature size [1]–[3]. In order to accommodate such uncertainty, conventional designs usually embed a large timing guardband into the design to ensure "always correct" operations. Such worst-case oriented design methodology, however, inevitably reduces the benefit provided by technology scaling.

One possible solution to tackle the timing uncertainty problem is to introduce timing error detection and correction mechanisms for error-resilient computing instead of error-free computing. *Timing Speculation* (TS) is a promising solution to achieve the above objective [4]–[7]. One representative TS technique is Razor [4], which conducts timing error detection with double sampling and restores the state of the system to a known-good pre-error state once detecting an error.

For circuits with timing speculation capability, the system performance/throughput can be further improved if we can optimize the circuit for timing error rate reduction. Various techniques (e.g., [8]–[14]) were proposed to achieve this objective. Most of them resort to techniques such as gate sizing and clock tuning to reduce timing error rates without altering circuit structure [8]–[12], limiting the flexibility of these solutions. In [13], [14], logic synthesis for timing speculation is performed. Without physical design information, however, the timing information at this stage is usually quite inaccurate. In this paper, therefore, we propose novel optimization methods for timing-speculated circuits using *Redundancy Addition and Removal* (RAR) technique, which iteratively removes target wires in the circuit and at the same time add alternative wires to keep circuit functionality. The advantages of the proposed solution include:

- With RAR technique, we are able to change circuit structure and hence have a high flexibility to manipulate circuit path delay distribution. To be specific, we have many options when targeting at wires on circuit critical paths that have high sensitization probability for removal, when compared to techniques such as gate sizing.
- RAR is performed after technology mapping in the design flow and hence we have more accurate circuit timing information when compared to those logic synthesis techniques.
- The hardware overhead of RAR-based technique is generally small as we conduct addition and removal of wires simultaneously.

The remainder of this paper is organized as follows. In Section II, we discuss related works and present the motivation of this paper. The proposed optimization framework for timing-speculated circuits based on RAR and an advanced algorithm are then detailed in Section III and Section IV, respectively. Next, Section V presents the experimental results on various benchmark circuits. Finally, Section VI concludes this paper.

II. Preliminaries

A. Timing Speculation

Timing speculation is a "better-than-worst-case" design technique that allows timing errors to occur and then detects and corrects them on-the-fly. By doing so, we can over-clock circuits to achieve higher throughput or scale down supply voltage for lower power consumption. Razor technique [4] is one of the most representative timing speculation techniques, which detects timing errors and conducts error recovery based on counterflow pipelining techniques [22]. During the error recovery phase, the pipeline is stopped for instruction replay.

In circuits with timing speculation capability, the overall throughput can be described as Eq. 1 shown [9]:

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

where T is the operational clock period, $P_e(T)$ is the timing error probability with respect to T, and r is the error penalty factor, indicating r clock cycles are needed to recover the system.

Based on the above, we can find that under a certain operational clock period T, system throughput is significantly affected by timing error probability $P_e(T)$, which motivates many optimization techniques of timing speculation (e.g., [8]–[10], [13]) to reduce system timing error probability. For example, Blueshift [8] proposes to identify and optimize the most frequently exercised critical paths with On-demand Selective Biasing (OSB) and Path Constraint Tuning (PCT), while DynaTune [9] assigns low threshold voltage V_t to the most dynamically critical gates for optimization. Both of these two methods lead to large power overhead, since the modified elements will result in extra leakage power consumption. In [10], the authors proposed to redistribute the path slack to increase the level of overscaling under the given timing error rate constraint. This strategy

increases the slack of frequently sensitized critical paths by enlarging the on-path gate size while decreases that of other paths. All the above methods can be treated as a post-processing method after the circuit structure is already finalized, whose effectiveness is hence limited. In [13], a logic synthesis technique is proposed with a different cost function. It is performed before technology mapping and accurate timing information is not available yet at that stage.

Motivated by the above, in this work we present a novel technique to optimize timing speculation based on redundancy addition and removal (RAR), which has the flexibility to modify circuit structure with accurate technology information available for better optimization.

B. Redundancy Addition and Removal

Redundancy addition and removal (RAR) technique, also known as rewiring, is a logic optimization technique that adds and/or remove wires in a circuit to optimize a certain objective with system function kept [15], [16]. Specifically, a logic network is optimized by iteratively adding and removing redundant wires that are identified by rewiring engine. Generally speaking, a rewiring engine can be regarded as a function blackbox, which uses the target wires under removal as input and outputs the corresponding alternative wires. The actions of target wire removal and alternative wire addition have to guarantee the original system function unchanged. In other words, the rewiring procedure is that you first add redundant wire by connecting a node to another without affecting the function of the circuit, and then some originally existing wire becomes redundant and removable.

Besides, it is also possible to remove given target wires by adding the corresponding alternative connections. The way to achieve this is to use mandatory assignments (MA) for the target wire's stuck-at fault test with automatic test and pattern generation (ATPG) applied. If the connection introduces conflicting assignments, an alternative connection is found.

Conventionally, RAR technique has been used to realize quite a lot of optimization objectives. In [15], [17], rewiring is used for a simplified logic with reduced hardware cost. In [18] an improved RAR technique is presented for circuit timing optimization. In [19], [20], the authors proposed to apply the ATPG-based RAR method to multi-level combinational logic circuit to optimize delay and power. In [21], the authors presented a novel framework based on RAR to reduce soft error rate. Different from prior works, in this paper we propose to use RAR technique to optimize timing-speculated circuits throughput by reducing system timing error probability. Our proposed methodology is performed after logic synthesis and technology mapping, as detailed in the following sections.

III. RAR for Timing-Speculated Circuits

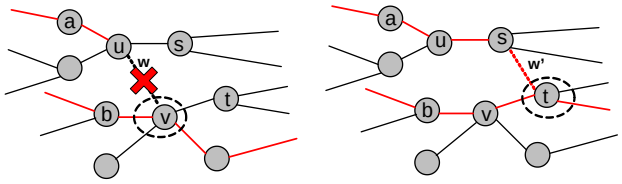
A. RAR Effects on Timing Speculation

By performing intentional wire removal and addition in a timing-speculated circuit, the circuit structure can be optimized to have less timing error probability and hence be more efficient from the perspective of timing speculation. Before getting into the details of our proposed RAR-based optimization technique, we are going to investigate the effects of RAR on timing speculation first. Generally speaking, there are two actions to take during the process of RAR: wire removal and wire addition. We define the wire for removal and the wire for addition as *target wire* and *alternative wire*, respectively. From these definitions, we can find that the so-called target wire and alternative wire are always discussed in pairs.

Definition 1: **Target wire** is the wire that is planned to be removed from a circuit.

Definition 2: **Alternative wire** is the wire that should be added into the circuit to maintain original circuit function after the corresponding target wire is removed.

An example of target wire removal is shown in Fig. 1(a). Assume wire w ($u \rightarrow v$) is a target wire for removal, gate u is its source node, and gate v is its destination node, it can be found that removing wire w has these impacts to system timing error probability: (i) All the critical paths that go through wire w will be broken and hence the system timing error probability contributed by these critical paths becomes non-existent; (ii) The masking effect from wire w on the paths that go through gate v is weakened and the timing error probability on these paths may be increased.



(a) Target wire removal. (b) Alternative wire addition.

Fig. 1. An example to show target wire and alternative wire.

As for alternative wire addition, as shown in Fig. 1(b), assume w' ($s \rightarrow t$) is an alternative wire of target wire w, gate s is the source node of wire w', and gate t is the destination node. we can conclude that adding alternative wire w' has these impacts on system timing error probability: (i) New paths are produced after w' is added and these paths may contribute extra timing error probability to the overall system; (ii) The masking effect from the newly-added alternative wire w' on the paths that go through gate t is strengthened so that the error contribution of these paths may be decreased.

B. Optimization Metric

By intentionally removing the wires that can break down critical paths, especially those frequently-sensitized critical paths , it is promising for us to improve timing performance and/or reduce timing error probability of circuits. From this viewpoint, how to obtain the error probability of critical paths and evaluate the impact of circuit structure changes on time error probability is quite interesting and important in our RAR-based optimization technique. Intuitively, we can simply resort to timing simulation to acquire timing error probability. For example, prior works (e.g., [13]) use timing simulation to acquire path sensitization probability, however this method is too time-consuming to be used in the iterative logic optimization procedure. To tackle this problem, a simple and effective optimization metric has been proposed in [14], wherein the probability for the delay of a certain path i to exceed clock period and the sensitization probability of this path are calculated to obtain the timing error probability as below [1]:

$$P_i = D_i \times S_i \quad . \tag{2}$$

Here, P_i is the timing error probability of path i, D_i is the probability for the delay of path i to exceed clock period, and S_i is the sensitization probability.

C. Optimization Algorithm

During our optimization process, the objective is to reduce timing error probability by wire removal and addition. We consider one

[1]Please refer to [14] for details.

target wire together with one of its alternative wires as a *pair*. In each optimization step, we *update/replace* one pair of wires, which means removing the target wire and adding the alternative wire. During this process, we expect the error probability reduction because of target wire removal can compensate the error probability increase from alternative wire addition, so that the overall error probability is reduced finally.

Path is a chain of gates and wires along which the signal can propagate from input to output and removing a wire can break all the paths through it. Here, we have an important observation that removing/adding a wire with more/less critical paths going through can contribute more in error probability reduction. Consequently, we propose to analyze the critical path distribution and estimate their error probability first. After that, we define a *weight* which indicates the importance of a wire during optimization process as follows:

Definition 3: The weight of a wire w (denoted as $weight(w)$) is defined as the sum of the error probability of all the critical paths that go through wire w.

It can be found that a wire with larger weight indicates more critical paths going through it and therefore removing this wire is expected to reduce more error probability. Meanwhile, since we have to add an alternative wire after target wire removal to remain circuit function, the key problem here is how to select a group of target wires with the largest weights as candidate target wires and then choose the best pair of wires to update the circuit. An intuitive way to tackle this problem is to try different target and alternative pairs and check their effects on error probability. To describe the importance of such a pair of wires, we define a metric called $benefit$ as below:

Definition 4: The benefit of a pair of wires, consisting of a target wire w and an alternative wire aw, is defined as the error probability reduction between the original circuit (denoted as "C") and the updated circuit (denoted as "$C-w+aw$").

Algorithm 1: RAR-based Optimization Algorithm

```
1  begin
2    while (max_benefit > 0) do
3      max_benefit = 0;
4      find all critical paths;
5      assign weight weight(w_i) for each wire;
6      sorted_wires=sort wires according to weight(w_i) from
       largest to smallest and select the group with top weights;
7      foreach w_i∈sorted_wires do
8        alt_wires = RewireEngine(w_i);
9        foreach aw_i∈alt_wires do
10         benefit =
           error_rate(C) − error_rate(C + aw_i − w_i);
11         if (benefit > max_benefit) then
12           max_benefit = benefit;
13           w = w_i;
14           aw = aw_i;
15         end if
16       end foreach
17     end foreach
18     if (max_benefit > 0) then
19       C = C + aw − w;
20     end if
21   end while
22 end
```

It is worth noting that, by calling the rewiring engine, we can obtain a number of alternative wires for each target wire. Among these pairs, each time only one pair is selected to update circuit structure, because the process of finding alternative wires is based on the analysis of original circuit structure, and if we update two pairs at the same time, it may cause wrong function. Updating two pairs at the same time can be considered as updating them one after another. When we are updating the second pair, the circuit structure has been changed because of the updating of the first pair. In that case, the equivalent relation between target wire and alternative wire may not stand any longer, and the second pair, found by rewiring engine based on the original structure, would violate circuit's original function.

Based on the above, we propose a basic version of optimization algorithm as demonstrated in Algorithm 1. Firstly, a structural analysis is performed to find out all critical paths in the circuit. Secondly, for each candidate wire w_i, we find out all critical paths that go through it and calculate its $weight(w_i)$. Next, we sort all the target wires in weight-descending order and select out those wires with largest weight (e.g. the top 50%). For each selected target wires w_i, rewiring engine is called to find out its alternative wires. Finally, for each pair of wires, we calculate its $benefit$ and select out the pair with maximum $benefit$ to update the circuit. This optimization procedure is repeated until we cannot obtain any $benefit$ any longer.

IV. Further Optimization with RAR

A. Structural Analysis

In the basic version of RAR-based optimization algorithm (see Algorithm 1), we only update one wire pair in each optimization step. After each step, rewiring engine has to read in and analyze the updated circuit structure again to find new target and alternative wire pairs. However, performing this kind of strucutural analysis is quite time-consuming. If more than one pair can be updated in each step, we can save a lot of time. Besides, in Algorithm 1, we choose wire pair to update circuit structure in $benefit$-descending order. However, this kind of greedy algorithm usually cannot guarantee global optimality because the global dependency information is not taken into consideration. That means, the order to update wire pairs will significantly affect the effectiveness but it is quite difficult to find out the best order because of the complicated dependency. From this point of view, if we can update more than one pair in each optimization step, at least the impact of optimization order can be weakened to some extent. Consequently, in this section we propose to utilize some rules through structural analysis to find out some sets of wire pairs that can be updated at the same time without violating original functionality.

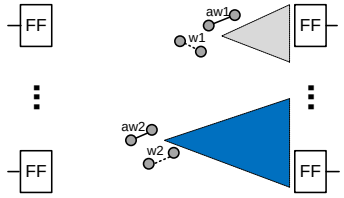


Fig. 2. An example of structural independent wire pairs.

Without loss of generality, let us consider a case with only two wire pairs in circuit structural analysis. As shown in Fig. 2, we have two wire pairs w_1-aw_1 and w_2-aw_2. Suppose their fan-out cones (the triangle areas in Fig. 2) have no overlapping with each other, we highlight this situation as "structural independent". When the two pairs are structural independent, updating one pair would not

affect the functionality around another pair. As a result, updating the two pairs without re-calling the engine will not violating the circuit functionality. Similarly, in the case of more than two pairs(pair A, B and C), if every two pairs (A and B, A and C, B and C) are mutually structural independent, we can update them at the same time.

B. Proposed Algorithm

To utilize the independency information obtained by structural analysis, we build up a graph as shown in Fig. 3. In this graph, the vertexes represent wire pairs and if two wire pairs are structural independent, they are connected with an edge. After that, we calculate the benefits of wire pairs and assign them to the vertexes as weights. Now the original problem to select a set of wire pairs that can be updated at the same time with largest error probability reduction becomes a new problem: how to find a clique in the graph so that the benefit sum of the vertexes in this clique is maximum.

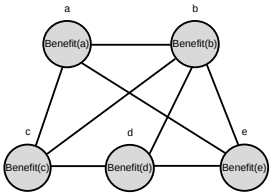


Fig. 3. A $benefit$-weighted graph.

Before directly investigating the above problem, it is inapplicable because the number of wire pairs inside a circuit can be quite large so that a huge amount of computation effort is needed to build and process such a graph. To reduce graph size and computation effort, we apply some rules to filter some wire pairs out of the vertexes candidates.

It is obvious that if the benefit (see Definition 4) of a wire pair is quite small or even negative, it must not be a good choice for circuit updating. Besides, since the error probability of a wire is closely related to the number of critical paths that go through it and the source/destination node of a wire can well describe it and its related critical paths, instead of calculating the wire's benefits, we investigate the rules for filtering out vertexes by defining a metric called a node's $criticality$ as follows:

Definition 5: The criticality of a node is defined as the sum of the weights of all the wires that serve as this node's fan-ins.

On the one hand, if there are many critical paths going through a node and a wire whose source node is this node (e.g. node s in Fig. 1(b)) is added into the circuit, it is probable that many critical paths will be introduced into the circuit. On the other hand, if there are few critical paths going through a node and a wire whose destination node is this node(e.g. node t in Fig. 1(b)) is added into the circuit, the critical path masking effect from the added wire would be quite small. From this point of view, when we add an alternative wire into the circuit, we prefer the wire whose source node is with less critical paths and destination node is with more critical paths. According to this observation, we build up our rules to filter out wire pairs during the process of graph construction:

- If the criticality of alternative wire's source node is larger than that of target wire's source node, this wire pair will not be included into the $benefit$-weighted graph;
- If the criticality of alternative wire's destination node is smaller than that of target wire's destination node, this wire pair will not be included into the $benefit$-weighted graph.

Algorithm 2: Advanced RAR-based Optimization Algorithm

```
1  begin
2      while (benefit > 0) do
3          find all critical paths;
4          assign weight weight(w_i) for each wire;
5          sorted_wires=sort wires according to weight(w_i) from
           largest to smallest and select the group with top weights;
6          foreach w_i∈sorted_wires do
7              alt_wires = RewireEngine(w_i);
8              decide aw_i and w_i pair filtered out or not;
9              foreach aw_i∈alt_wires do
10                 benefit =
                   error_rate(C) − error_rate(C + aw_i − w_i);
11                 if (benefit > 0) then
12                     add pair P(w_i, aw_i)to the graphG;
13                     assign benefit to vertex P(w_i, aw_i);
14                 end if
15             end foreach
16         end foreach
17         assign edges between different vertex P in graph G;
18         P_best(w, aw) = max_weight_clique(G);
19         benefit = error_rate(C) − error_rate(C + aw − w);
20         if (benefit > 0) then
21             C = C + aw − w;
22         end if
23     end while
24 end
```

With the $benefit$-weighted graph constructed, we present our advanced RAR-based optimization algorithm as shown in Algorithm 2. Firstly, we perform structural analysis to find out all the critical paths and calculate wire weights. Secondly, we construct a $benefit$-weighted graph with each vertex representing a wire pair, assign the benefits of wire pairs as the weights of vertexes, and connect the structural independent vertexes with edges. After that, we try to find the clique with largest sum of $benefit$-weights. This process is repeated until there is not any error probability reduction that can be obtained any longer.

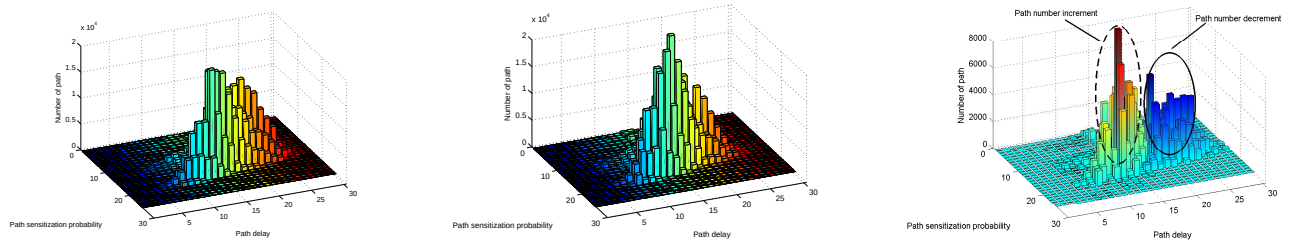
V. Experimental Results

A. Experimental Setup

To evaluate the effectiveness of RAR-based optimization technique, we conduct experiments on *ISCAS'89* benchmark circuits with UMC's 130nm technology and perform Monte Carlo simulation to study the impact of variation effects (we use Gaussian distribution with standard deviation 10%). To get timing error probability, we perform simulation with random inputs in our experiments and each simulation is performed for 100,000 cycles, which 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]. We sweep the operational clock period for each case to find out the best one with largest throughput calculated according to Eq. 1.

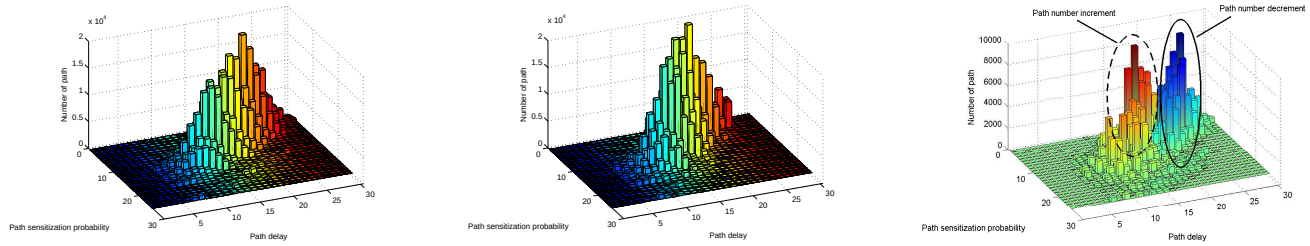
B. Results and Discussion

First of all, we present the throughput improvement and hardware cost in Table I. $RAR_{intuitive}$ represents the results of basic RAR-based optimization algorithm and $RAR_{advanced}$ represents the results of advanced RAR-based optimization algorithm. Column 2 shows



(a) Original (b) Rewire (c) *Comparison*

Fig. 4. Path delay distribution and sensitization probability on *s*35932.



(a) Original (b) Rewire (c) *Comparison*

Fig. 5. Path delay distribution and sensitization probability on *s*38584.

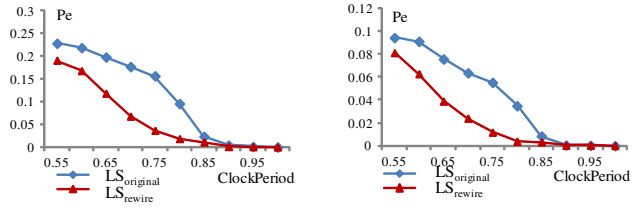
TABLE I
RESULTS ON SYSTEM THROUGHPUT AND HARDWARE COST.

Circuit	Throughput(MHz)					Hardware				Runtime($Sec.$)	
	Original	$RAR_{intuitive}$	$\Delta_1(\%)$	$RAR_{advanced}$	$\Delta_2(\%)$	*Original*	$RAR_{intuitive}$	$RAR_{advanced}$	$\Delta_2(\%)$	$RAR_{intuitive}$	$RAR_{advanced}$
s298	1071.53	1185.98	10.68	1210.50	12.97	681	671	681	0	0.83	0.71
s344	488.56	564.32	15.51	576.24	17.95	796	792	790	-0.75	1.27	1.44
s349	488.56	564.32	15.51	579.73	18.66	794	796	800	0.76	1.41	1.54
s382	464.32	581.91	25.32	589.14	26.88	1101	1087	1082	-1.73	1.76	1.71
s386	380.73	474.60	24.65	481.83	26.56	1273	1273	1273	0	1.72	2.04
s444	575.12	605.45	5.27	635.63	10.52	1252	1255	1243	-0.72	4.42	4.96
s526	725.83	785.69	8.25	827.00	13.94	1094	1084	1090	-0.37	8.32	7.64
s641	230.10	252.82	9.88	262.99	14.23	1311	1311	1311	0	12.76	13.00
s713	172.14	196.84	14.35	208.08	20.88	1377	1377	1377	0	15.92	15.01
s953	605.81	738.51	21.90	812.36	34.10	2177	2184	2183	0.28	8.13	9.03
s1196	321.75	384.93	19.64	396.22	23.15	1795	1797	1795	0	25.84	23.00
s1238	313.45	354.35	13.05	375.06	19.66	1685	1685	1701	0.95	33.91	36.02
s5378	354.83	406.50	14.56	426.65	20.24	14169	14172	14174	0.04	34.86	39.05
s9234	245.45	286.05	16.54	295.86	20.54	19575	19580	19578	0.02	133.62	128.22
s13207	190.68	220.76	15.78	246.87	29.54	37025	37137	37392	0.99	480.23	492.14
s35932	285.98	322.20	12.67	382.44	33.73	114320	114146	114218	-0.01	1593.02	1655.28
s38584	115.91	124.80	7.67	139.85	20.65	77747	77747	77714	-0.04	4628.28	5325.45
AVERAGE			14.78		21.42				-0.03		

Δ_1: Difference between *Original* and $RAR_{intuitive}$;
Δ_2: Difference between *Original* and $RAR_{advanced}$.

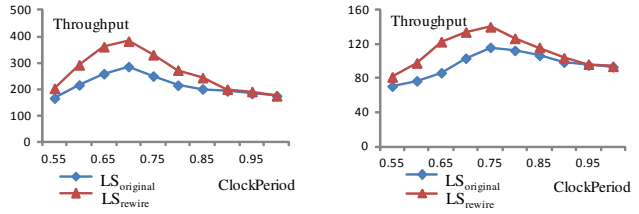
the throughput results for the original benchmark circuits. Column 3 shows the throughput results optimized by $RAR_{intuitive}$ and Column 4 indicates the improvement ratio of $RAR_{intuitive}$, from which we can observe a 14.78% throughput improvement. In Column 5, the throughput optimized by $RAR_{advanced}$ is shown and Column 6 shows its improvement ratio compared with the original one. On average 21.42% throughput improvement can be obtained. Column 7-9 demonstrate the circuit area of the original circuit, the circuit optimized by $RAR_{intuitive}$ and the circuit optimized by $RAR_{advanced}$. In Column 10, the circuit area difference ratio between the original and $RAR_{advanced}$ is shown. We can see that the hardware cost is usually very small and sometimes negative. Column 11 and Column 12 demonstrate the runtime of $RAR_{intuitive}$ and $RAR_{advanced}$, respectively. By comparing the results in these two columns, we can see their runtime difference is not large. This is because, although a maximum clique algorithm is needed in the advanced algorithm, the calculating round is reduced as multiple wires are selected in each round.

To better demonstrate the effectiveness of our proposed technique, we take *s*35932 and *s*38584 as examples to show the impact of our optimization on path delay and sensitization probability distribution of the circuit. In Fig. 4 and Fig. 5, x-axis indicates the path delay, y-axis shows the path sensitization probability and z-axis represents the number of paths. Fig. 4(a)(Fig. 5(a)) and Fig. 4(b)(Fig. 5(b)) show the path distribution of the original circuit and $RAR_{advanced}$, respectively. Fig. 4(c) and Fig. 5(c) demonstrate the difference on



(a) s35932 (b) s38584

Fig. 6. Timing error probability comparison.



(a) s35932 (b) s38584

Fig. 7. Circuit throughput comparison.

path distribution between them. The red bars within the dashed circle represent the increment of path number while the blue ones within the solid circle represent the decrement of path number after $RAR_{advanced}$. From this comparison, the number of the paths with smaller delay or smaller sensitization probability is increased while the number of the paths with larger delay or larger sensitization probability is decreased, which justifies the decrease of timing error probability after using our technique.

Finally, we also show the changes of timing error probability (Fig. 6) and circuit throughout (Fig. 7) of the example circuits *s*35932 and *s*38584 with respect to different clock periods. From Fig. 6 we can see that in most cases, the timing error probability is cut down after using the rewiring technique. In Fig. 7, we can see the throughput increment in most cases and the peak of the curve is the selected operational clock period.

VI. Conclusion

In this paper, timing speculation optimization techniques based on RAR are proposed. When compared to existing solutions, our technique has the advantages of both flexibility to change circuit structure and accuracy of detailed technology information. Results show the timing error probability is largely cut down, which improves the performance for timing-speculated circuits.

VII. Acknowledgement

This work was supported in part by the Hong Kong SAR Research Grants Council under General Research Fund No. CUHK418111 and No. CUHK418812.

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. 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.

[4] 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.

[5] 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.

[6] T. Liu, and S. lien Lu, "Performance improvement with circuit-level speculation," in *Proc. International Symposium on Microarchitecture*, 2000, pp. 348–355.

[7] T. M. Austin, "Diva: A reliable substrate for deep submicron microarchitecture design," in *Proc. International Symposium on Microarchitecture*, 1999, pp. 196–207.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] Y. Liu, et al. "On logic synthesis for timing speculation," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 591–596.

[15] K. Cheng; L. Entrena,"Multi-level logic optimization by redundancy addition and removal," European Design Automation Conference, pp. 373-377, 1993.

[16] S. Chang, G. van, M. Sadowska,"Fast Boolean optimization by rewiring," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp.262–269, 1996.

[17] S. Chang, G. van, M. Sadowska, "Circuit optimization by rewiring," IEEE Transactions on Computers, vol.48, no.9, pp.962-970, Sep 1999.

[18] L. Entrena, J. Espejo, E. Olias, J. Uceda, "Timing optimization by an improved redundancy addition and removal technique," Design Automation Conference at Europe, pp.342-347, 1996.

[19] A. Veneris, "Logic rewiring for delay and power," IEEE International Symposium on Circuit and Systems, 2002, nov.

[20] M. Amiri, A. Veneris,I. Ting, "Design rewiring for power minimization [logic design]," IEEE International Symposium on Circuits and Systems, vol.4, pp. IV-305–IV-308, 2002.

[21] K. Wu and D. Marculescu, "Soft error rate reduction using redundancy addition and removal," Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific, March, 2008, pp.559-564.

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

[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. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010 , pp.487-496.