Re-Synthesis for Cost-Efficient Circuit-Level Timing Speculation

Yuxi Liu[†], 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: {yxliu,fyuan,qxu}@cse.cuhk.edu.hk

ABSTRACT

As the transistor feature size is continuously scaled down, integrated circuits are more vulnerable to process, voltage and temperature (PVT) variations, causing infrequent timing errors. Various techniques have been proposed to tackle this problem and circuit-level timing speculation is one of the most promising solutions. However, directly applying such technique can be quite costly in terms of area overhead and energy consumption. In this paper, we propose cost-efficient re-synthesis solutions to tackle this problem. We try to reduce the number of suspicious flip-flops (FFs) that might have timing errors by retiming techniques, which relocate some suspicious FFs without increasing critical path delay. An efficient and effective algorithm is then utilized to pad those short paths linking the remaining suspicious FFs to ensure the functional correctness of timing speculators. Experimental results show that the proposed solution can achieve significant area reduction for timing speculation.

1. INTRODUCTION

With the continuous downscaling of transistor feature size, there is an increasing uncertainty for the behavior of today's integrated circuits (ICs), often exhibiting as infrequent timing errors that occur on critical (or near-critical) paths [1, 2]. There are multiple factors that contribute to this effect. Inevitable process variation leads to the mismatch of timing performance between the designed value and the actual one [3]. Runtime environmental fluctuations (e.g., temperature variation, crosstalk, and IR-drop), and aging effects such as hot carrier injection (*HCI*) and negative-bias temperature instability (*NBTI*) all possibly cause timing errors [4]. Traditional design methodology requires designers to embed considerable design guard band to prevent any timing failure, which, unfortunately, diminishes the benefit of technology scaling.

Circuit-level timing speculation (e.g., the well-known Razor technique [5]), being able to detect timing errors at online stage, is a very popular technique that can react to the error quickly and recover from it by rolling back to a known-good pre-error state. There are many different types of timing error detector designs presented in the literature (e.g., [6, 7, 8, 9, 10, 5]), and most of them are based on double-sampling of the input to the flip-flops (FFs) driven by critical paths, denoted as *suspicious FFs* in this paper. By combining circuit-level timing speculation with dynamic management schemes such as dynamic voltage/frequency scaling (*DVFS*) and adaptive body biasing (*ABB*), we can achieve better tradeoff among performance, energy consumption and reliability of the circuit[11, 12, 13]. In addition, such technique is also very helpful for post-silicon debug as we are able to find the root cause of the timing errors easily.

However, the above benefits are not obtained for free. All *suspicious flip-flops* that are vulnerable to timing error need to be equipped with timing speculators. Furthermore, a certain period (or timing window) right after the clock edge is set to monitor late transition on suspicious FFs for timing speculation. Any occurrence of transition in this timing window is regarded as a timing error. It is therefore important to distinguish the late coming transition on critical paths from that on short paths to guarantee the correctness of error detection. Consequently, the delay of combinational paths that end up with a suspicious FF must be longer than the timing window. This is achieved by padding short paths with buffers, which also contribute to hardware overhead of the circuit-level timing speculation techniques.

Due to the above, the hardware cost for timing speculation can be quite high. To the best of our knowledge, however, no existing work has considered to optimize the circuit for timing speculation. In this paper, we propose cost-efficient re-synthesis solutions to tackle this problem. Our proposed technique can be divided into two steps. In the first step, we try to reduce the number of suspicious FFs by retiming the circuit [14]. It is important to note that, since the optimization objective is different from that of existing retiming techniques for circuit delay, area, power and even testability optimizations (e.g., [14, 15, 16]), essentially we can take a circuit after retiming for performance and apply the proposed technique to reduce the number of suspicious FFs. Then, for the remaining suspicious FFs, the second step is to pad short paths connecting to them in such a manner that buffers are inserted and shared among short paths whenever possible to reduce hardware overhead. In terms of design flow, the proposed re-synthesis step can be conducted after the traditional logic synthesis for timing optimization step and followed by physical design. Although the estimation for critical path delay cannot be very accurate at this stage, it is sufficient to our optimization objective since we can consider the worst case and conservatively leave a margin during the re-synthesis process.

The remainder of this paper is organized as follows. Section 2 presents the preliminaries and motivates this work. In Section 3 and Section 4 , we detail the proposed techniques for the reduction of suspicious FFs and the algorithms for padding short paths, respectively. Experimental results on various benchmark circuits are then presented in Section 5. Finally, Section 6 concludes this paper.

2. PRELIMINARIES AND MOTIVATION

2.1 Circuit-Level Timing Speculation

Various techniques have been presented for online timing error detection [6, 7, 8, 9, 10, 17]. Without loss of generality, let us consider the representative *Razor* flip-flop [5] to demonstrate how timing error detectors work. As shown in Fig. 1, a Razor FF contains a main flip-flop, an additional shadow latch and some control logic. The main flip-flop latches the output signal at the clock edge with possible timing error, while the shadow latch, controlled by a delayed clock signal,

DAC 2011, June 5-10, 2011, San Diego, California, USA.

latches the signal a fraction of a cycle later, which guarantees to receive the correct value. Consequently, when the shadow and the main FF values do not agree, indicated by the comparator, the timing error is detected. To make use of this timing speculation technique, it is necessary to replace all suspicious FFs with Razor FFs (or other timing speculators).

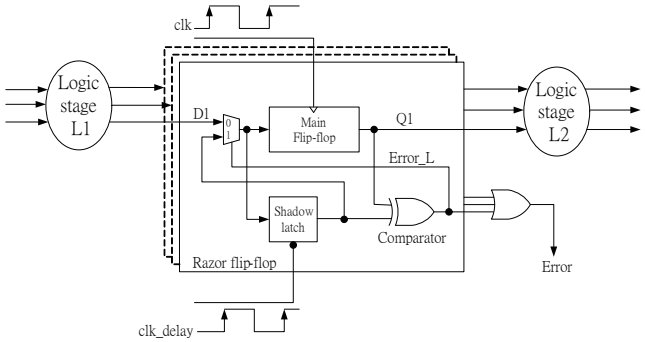

Figure 1: Schematic of Razor flip-flop

One important issue in Razor-like timing speculation techniques is the so-called *short path padding* problem, as shown in Fig. 2. As discussed earlier, the timing error is detected by re-sampling the output at a specified period later than the clock edge and comparing it with the original output in [5]. However, if the data, which should be the result of the next clock cycle, comes too early through short paths and arrives at the shadow latch before the rising of the delayed clock, it will overwrite the previous data that should be stored in the shadow latch. Under such situation, the Razor FF may make a wrong judgement on whether there is a timing error. As a result, in order to guarantee the correct function of the timing error detector, the min-delay path to the suspicious FFs must be long enough to satisfy the hold time constraint of the delayed clock, so as to avoid data of the next clock cycle overwriting what is stored in the shadow latch. Therefore, extra effort should be paid to pad these short paths connected to suspicious FFs.

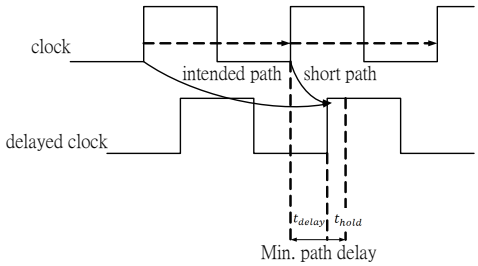

Figure 2: Short path problem

In [5], short path padding is conducted by simply adding the short path constraints during logic synthesis. In [18], the author proved the possibility of padding short paths with suitable delay of buffers and proposed two methods to pad the short paths to cut down the padding cost. One is a simple greedy heuristic approach and it is not so efficient. The other method is based on linear programming, which is quite time-consuming. In [6], the author padded each chosen *node* along the short paths. The amount of padding buffers for each node is determined by the so-called *slack* value constrained by the length of the longest path and the so-called *desired* value constrained by the shortest path through the certain node. In addition, the padding order is based on *slack* and *desired* values, which is not efficient considering the cost. Taskin *et al.* [19] proposed a delay insertion method at the sequential block level, however, how to do that at the combinational block level is not clear. In [20], although the author proposed a more efficient delay insertion method with linear programming, it also suffers from the same problem with other LP based methods. In [21], Mitrajit *et al.* proposed to initially assign buffers in topological order and use a retiming like technique to reduce the cost. While interesting, the buffer sharing is not considered and it is still time- and space-consuming.

2.2 Motivation

The hardware cost for timing speculation can be illustrated in the following equation:

$$Cost = Num_{sus\_ff} \times Cost_d + Cost_{pad}(Num_{sus\_ff}) + Others \quad (1)$$

The first item indicates the hardware cost for Razor FFs, in which Num_{sus_ff} and $Cost_d$ denote the number of suspicious FFs and extra hardware cost of a single Razor FF respectively. Since only the short paths that connect to suspicious FFs need to be padded, the hardware cost of padding buffers $Cost_{pad}$ is a function of the number of suspicious FFs Num_{sus_ff}. This is what the second item represents in equation 1. The cost of additional routings and a global controller of the error detection scheme is regarded as the third item *Others* in equation 1. The aforementioned different error detection techniques themselves can only affect the factor $Cost_d$, which has a very limited impact on the total *Cost*.

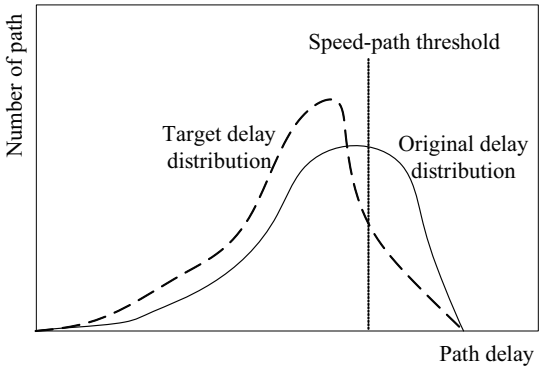

Figure 3: Path delay distribution after logic synthesis.

Conventional logic synthesis tool mainly targets to reduce the timing delay of the most critical path(s) in the circuit, and the path delay distribution after optimization may look like the solid line shown in Fig. 3. If, however, we can resynthesize the circuit with the distribution as the dotted line shown in Fig. 3, the number of suspicious FFs could be significantly reduced without affecting the longest delay of the circuit. This can be achieved because some of the less critical paths (but could have timing errors with timing speculation) are not optimized during conventional logic synthesis, and we may be able to reduce their delays so that timing errors would not occur on them with retiming techniques. By doing so, Num_{sus_ff} is smaller and it also facilitates to reduce $Cost_{pad}$, and hence the total hardware cost for timing speculation can be cut down. In addition, we would like to further reduce the padding cost by sharing buffers whenever possible.

The above has motivated the proposed resynthesis techniques for cost-efficient timing speculation.

3. REDUCING SUSPICIOUS FFS BY RETIMING

For a particular suspicious FF, if we are able to change those critical paths that drive it to be non-critical, this FF will be no longer suspicious. The is achievable with the help of retiming technique. That is, to shorten the propagation delay on a particular path, we can move backward the critical path's receiving FF towards the path's starting point and/or move forward its sending FF towards the ending point (see Fig. 4). This, however, may increase the delay on other paths and hence it is important to consider related paths together.

Mixed integer linear programming (MILP) based approaches for retiming [14, 15, 22, 23] can be used for suspicious FF reduction, by simply changing the objective function in the mathematical programming model. However, this method is not scalable with the increase of circuit size, and at the same time, retiming FFs globally may change the circuit structure a lot, which is not desirable from designers' viewpoint. Hence, we develop an efficient and effective heuristic method

for suspicious FF reduction, without heavy impact on the circuit structure, as detailed in this section.

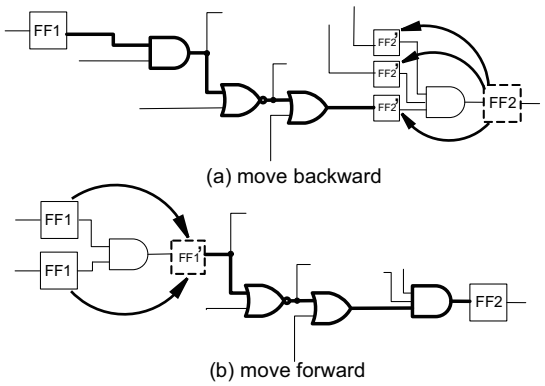

Figure 4: Relocating flip-flops.

3.1 Suspicious Flip-Flops

To reduce suspicious FFs, we mainly consider to relocate two types of FFs. One is *backward FF*, by which moving backward this FF can help to shorten the critical paths. The other is *forward FF*, by which moving forward the FF is helpful to reduce the length of critical path. Obviously, this kind of FF is in the fan-in logic cone of suspicious FFs and should satisfy forward moving condition. There are also some FFs that serve as both backward FF and forward FF at the same time and we call it *bidirectional FFs*. That is, for this type of FF, if we move them backward, it can help to eliminate the suspicious FF in its fan-in cone or change itself from suspicious to non-suspicious while if moving forward, it can be helpful to eliminate suspicious FF in its fan-out cone. Since the benefits of moving backward and forward of these FFs are likely different, the order of moving bidirectional FFs may lead to different results. On the contrary, for those mere backward FF or mere forward FF, referred to as *unidirectional FF*, since they can only move towards a single direction, the moving order has small impact on the result.

From the above, we divide this algorithm into two steps. The first step is to deal with those suspicious FFs in whose fan-in cone or fan-out cone there are no bidirectional FFs (we call *unidirectional suspicious FFs*). The second step is to tackle the remaining bidirectional suspicious FFs which connect to bidirectional FFs in a specific order.

3.2 Relocating Unidirectional Suspicious FFs

Dealing with these unidirectional suspicious FFs, we first check out all forward FFs in their fan-in cone and move them forward (like Fig. 4 (b)), which can help to shorten the critical paths to the current suspicious FF without affecting other suspicious FFs. This movement is stopped until the following situations happen:

- Moving condition cannot be met, which means a FF can move from the input to output only when there are FFs in all the gate's inputs;
- The forward FF becomes suspicious FF;
- The length of related critical path below the threshold value;

Then, we move backward these unidirectional suspicious FFs one by one to further shorten critical paths (like Fig. 4 (a)). In this way, paths to them can be shortened while those from them will be lengthened. As a result, this movement may make new suspicious FFs in its fan-out cone after we eliminate the current suspicious FF. So if we repeat this movement for the next fan-out cone's FFs, the critical paths will move towards the following sequential levels. During the process of critical paths' moving towards primary output, they may be eliminated in some short path level. After this step, we can eliminate part of the unidirectional suspicious FFs. Figure 5 details how FFs moves backward and it is illustrated as follows.

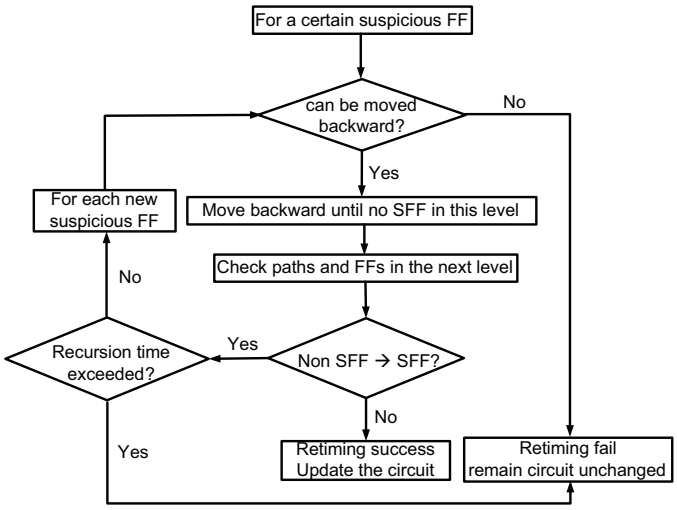

Figure 5: Moving backward flow.

Before moving one suspicious FF backward, we first check whether this FF can move to its preceding gate's input. Only when all fanouts of the preceding gate are connected with FFs can it move back. Otherwise, we just skip the current suspicious FF and turns to another. When a suspicious FF moves from the output to input of one gate, it may produce more than one FFs according to the input pins. Hence we continue to move each of the new FFs until none of the new FFs produced by the current suspicious FF are still suspicious as shown in figure 5.

Because moving FF backward will lengthen the paths starting from it, we check the timing of FFs in the fan-out cone to see whether there is non-suspicious FF becoming suspicious. If not, it indicates moving backward the current suspicious FF succeeds and we are able to eliminate it. While on the contrary, we continuously move backward those new suspicious FFs and try to eliminate them. The procedure is just the same but here we set up a recursion time limitation. When exceeding the restraint but there is still suspicious FF, it indicates relocating current suspicious FF fails. The original circuit are restored and we turn to another. The above process is repeated for every unidirectional suspicious FF in the circuit.

3.3 Relocating Bidirectional Suspicious FFs

After the first step, unidirectional suspicious FFs are removed if possible. Then we turn to the other type of so called bidirectional suspicious FFs. For this kind of suspicious FF, the order of moving can largely affect the results. As a result, we need to relocate these FFs according to a specific order to get a better result as shown in Fig. 6.

For a certain bidirectional suspicious FF, we initially find out all its related bidirectional FFs (see Fig. 6). Then for each of the related bidirectional FF, the gains of moving backward and forward are calculated in order to know which direction it should move towards and which suspicious FF we should first tackle.

The gains can be estimated as follows. When a FF moves backward, it can potentially benefit the suspicious FFs in its fan-in cone. It is because when it moves backward, the suspicious FF in the fan-in cone will have more flexibility to move backward and shorten its related critical paths without affecting the following level. So here the gain is defined as the number of suspicious FFs in its fan-in cone that can be potentially benefited. Similarly, when a FF moves forward, it may potentially benefit the suspicious FFs in its fan-out cone because the critical paths connected to these suspicious FFs may be shortened. The gain is also defined as the number of suspicious FFs in its fan-out cone potentially being benefited.

Then we find out the largest gain and get the direction of this gain. If the largest gain is from the bidirectional FF's backward moving in the fan-out cone or forward moving in the fan-in cone of current suspicious FF, it means the direction is towards current suspicious FF and so it is

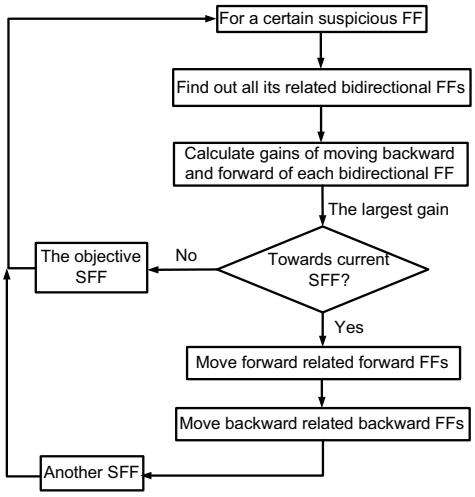

Figure 6: Moving bidirectional suspicious FF flow.

the objective FF. So we can try to remove it. Moving forward the FFs in fan-in cone, backward itself and FFs in its fan-out cone may help to eliminate it. After that, we turn to another suspicious FF and repeat the above steps. While in the other cases, the objective FF is not the current suspicious FF, we just turn to the objective FF first and repeat the steps as above leaving the current suspicious FF tackled later (see Fig. 6).

Obviously, moving FFs backward may increase the total number of FFs even if successfully removing a particular suspicious FF. In this case, although we save the cost of a Razor FF and the padding buffers, the cost of additional FFs may counteract the benefits. For this reason, during this process, we also balance the cost of additional FFs and benefits from removing suspicious FFs. Only when there is net benefit will we actually do it.

Using this method, the number of suspicious FFs is cut down so that the overhead, both from assigning Razor FF and padding short paths can be reduced. What we should do next is to pad the short paths.

4. PADDING SHORT PATHS

When padding short paths linking suspicious FFs, our objective is to insert buffers into the circuit in order to lengthen these paths to exceed a certain threshold value. A straightforward method is to simply rely on the synthesis tool to tackle this problem. We can add the minimum path delay constraints from particular inputs to outputs during logic synthesis to achieve short path padding. However, considering that we need to pad all the short pathes leading to suspicious FFs, without a global view, such a path-by-path solution may suffer from more hardware cost, thus simply relying on the synthesis tool is not enough for efficiency. Therefore, in the following subsections, a mathematical programming method is introduced briefly and we also propose a more efficient heuristic method to solve this problem.

4.1 Linear Programming for Padding

To solve the padding problem, linear programming (LP) approach proposed by [18] can give a good solution. For each gate in the circuit, the early arrival time of gate i is modeled as a_i and the late arrival time of gate i is modeled as A_i. Edge e_{ji} represents the connection from the output of gate j to gate i and w_{ji} shows the delay inserted on the edge e_{ji}. d_{ji} demonstrates the interconnect delay of edge e_{ji}. The arrival times at gate i are computed as:

- If i's input is flip-flop or primary input, then a_i and A_i are specified as the delay of the interconnect;
- Otherwise,

$$a_i = min\{a_j + w_{ji} + d_{ji}\} \quad j \in FI(i) \tag{2}$$

$$A_i = max\{A_j + w_{ji} + d_{ji}\} \quad j \in FI(i) \tag{3}$$

Therefore, we can consider the following relaxed linear programming to pad the short paths:

$$min(\sum w_{ji}) \quad w_{ji} \in E$$

$$a_i \le a_j + w_{ji} + d_{ji} \quad \forall j \in FI(i) \tag{4}$$
$$A_i \ge A_j + w_{ji} + d_{ji} \quad \forall j \in FI(i) \tag{5}$$
$$A_i \le T \quad \forall i \in PO \tag{6}$$
$$a_i \ge T/2 \quad \forall i \in PO \tag{7}$$
$$A_i = w_{ji} + d_{ji} \quad \forall j \in PI \tag{8}$$
$$a_i = w_{ji} + d_{ji} \quad \forall j \in PI \tag{9}$$

For similar reason as above, linear programming method is not applicable to a large scale circuit since it will become too time- and space-consuming. Hence, a heuristic method is needed. The following introduces the proposed heuristic method.

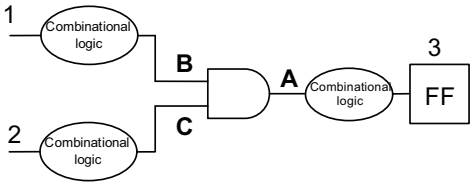

Figure 7: Impact of different padding orders.

4.2 Heuristic Method for Padding

We notice that the order of padding in different positions may have a large impact on the cost. Because there are usually a large number of short paths connecting to one suspicious FF, some short paths might share the same positions. Padding buffers in the shared part benefits cost-efficiency. Therefore, padding order leads to a great difference in overhead. For example, in Fig. 7, supposing path $1 \rightarrow 3$ and path $2 \rightarrow 3$ are short paths and their length are both 4 units, the timing constraint requires no shorter than 5 units, and a single buffer causes 1 unit delay. If we first add buffer to position B to increase delay of path $1 \rightarrow 3$, we also need to pad path $2 \rightarrow 3$ in the same way and 2 buffers are consumed in this padding scheme. However, if we first insert one buffer to position A, both two short paths could be well padded, which costs half.

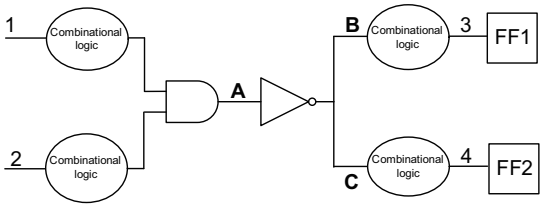

Figure 8: Padding short paths globally.

As a result, how to determine an order to achieve highest cost-efficiency becomes critical. Many positions are shared by multiple short paths and buffers inserted in a position can lengthen delays of all short paths sharing it at the same time. Therefore, padding in the positions shared by more short paths will cause a lower cost while achieving the same effect. So if assigning priority to positions shared by more short paths, padding may succeed costing a lower overhead. As a result, priority of padding in a position should be based on how many short paths the position is shared by.

In addition, we consider the short paths connecting to all suspicious FFs globally. In Fig. 8, path $1 \rightarrow 3$ and $2 \rightarrow 4$ are both short paths needing to be padded, and FF1 and FF2 are both suspicious FFs. If short

paths connecting to FF1 and FF2 are padded one after another, the position A is regarded to be occupied by only one particular short path respectively. So the position A has the same priority with B or C while padding. However, if all short paths associated with both FF1 and FF2 are considered simultaneously, position A should have a higher priority than either B or C, because it is shared by two short paths under this situation.

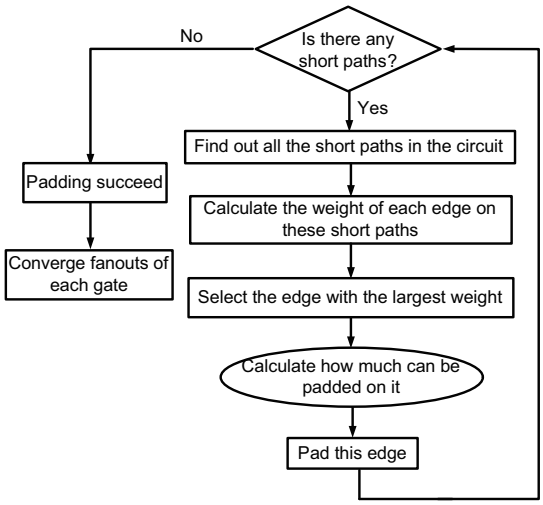

Figure 9: Short paths padding flow.

The proposed padding process is shown in Fig. 9. We first check whether there is short path connecting to any suspicious FFs. If there is no short path, the padding process ends. If any short paths exist, padding is conducted in a particular order according to their weights, which is determined by the amount of sharing short paths. So as shown in Fig. 9, we then find out all the short paths in the circuits. After that, the weights of edges on these short paths are calculated. Then the edge with the largest weight is selected out to pad. The delay that can be padded on one edge should be decided, which is determined with two constraints. Firstly, padding should not affect the corresponding allowed largest path delay. In Eq. 10, D_{slack} defines the delay slack to the allowed largest path delay D_{upper}. As there may be multiple paths going through this edge, $max(D_{p_i})$ denotes the largest delay of these multiple pathes.

$$D_{slack} = D_{upper} - max(D_{p_i}) \tag{10}$$

The second constraint is to make the short paths longer than the threshold value. $D_{desired}$ in Eq. 11 indicates how much extra delay need to be added to make short paths meet timing constraint. $D_{threshold}$ denotes the required lowest delay for any path connecting to a suspicious FF. Similarly, $min(D_{p_i})$ denotes smallest delay of the paths going through the current edge.

$$D_{desired} = D_{threshold} - min(D_{p_i}) \tag{11}$$

Extra delay padded on one edge should be no more than min(D_{slack}, $D_{desired}$). Considering the constraints, aforementioned operations are repeated then until there is no short path.

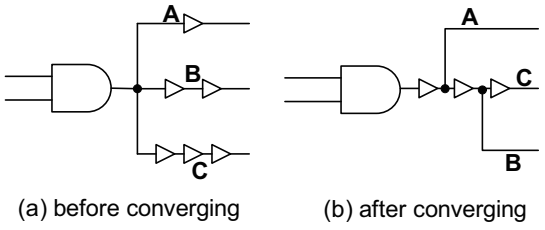

Figure 10: Examples for merging fanouts.

After padding stops, we proceed to further cost reduction. We merge the buffers inserted on the edges that belong to fanouts of one single gate. Example is shown in Fig. 10. Before merging, we insert one buffer on edge A, two buffers on edge B and three buffers on edge C. As they all are fanouts of the same gate, we can merge buffers on these edges so that they can share and decrease the number of buffers needed if ignoring the change of buffer's driven ability.

Table 1: Relocating Results Comparison

Circuit	Proposed method			Mathematical programming		
	SFF # before	SFF # after	Runtime	SFF # before	SFF # after	Runtime
s208	2	2	0.1s	2	2	1m
s298	5	4	0.3s	5	3	2m
s344	3	2	0.2s	3	2	13m
s349	3	2	0.2s	3	2	12m
s382	10	7	0.7s	10	5	17m
s386	5	3	0.5s	5	2	20m
s444	10	6	0.6s	10	4	32m

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

Experiments are performed on *ISCAS89* and *IWLS05* benchmark circuits. We set the threshold value determining suspicious FF to be 80% of the maximum path delay while short paths with less than 50% of the maximum path delay connecting to suspicious FFs need to be padded.

During timing analysis, we consider both the gate delay and interconnect delay. The gate delay information is from *FSC0G_D 0.13um* Standard Cell. The interconnect delay is assumed to be proportional to the distance between two gates after placement [22], which is generated by the open source Capo 10.5 placer [24]. For linear programming solver, we use MATLAB with YALMIP toolbox.

To show the practicability of our methods, we apply our re-synthesis techniques after the circuits have been optimized for timing. That is, we synthesize and optimize the benchmark circuits using a commercial synthesis tool first, which has already try to balance sequential logic for performance improvement.

5.2 Experimental Results

First, to show the efficiency of our FF relocation heuristic, we compare the number of suspicious FFs before and after retiming by using both mathematical programming (derived from [14] and [22]) and the proposed heuristic method for some circuits in Table 1. Column 2, 3, 5 and 6 show the number of suspicious FFs before/after retiming by these two methods and column 3 and 6 represent the runtime respectively. From the result, we can see that the result of our heuristic method is only a little worse than the mathematical programming method but it can solve the problem much faster. Here we only show the results of some small circuits. It is because it is too time- and space-consuming for large circuit and also the scale limit of the tool used.

Table 2: Padding Results Comparison

Circuit	SFF # before	SFF # after	Proposed method			LP method [18]		
			Buffer # before	Buffer # after	Runtime	Buffer # before	Buffer # after	Runtime
s208	2	2	9	9	0.5s	9	9	15s
s298	5	4	27	22	0.9s	24	20	30s
s344	3	2	36	29	1.1s	35	26	2m
s349	3	2	30	22	1s	25	20	2m
s382	10	7	28	21	1.3s	27	21	4m
s386	5	3	22	14	1.2s	21	12	3m
s444	10	6	29	20	1.6s	27	17	5m

Table 2 indicates the comparison of padding results before and after relocating using both the linear programming method and proposed heuristic method. Column 2 and 3 show the number of suspicious FFs before and after relocating FFs. Column 4, 5, 7 and 8 represent the cost

Table 3: Benefits From Re-synthesis

Circuit	Original area	Before relocating			After relocating			Reduced cost(%)
		# of SFF	Total area	Extra cost(%)	# of SFF	Total area	Extra cost(%)	
s641	1291	8	1637	26.8	6	1549	19.9	25.4
s713	1243	8	1614	29.8	7	1525	22.7	23.8
s953	2951	11	3650	23.7	8	3415	15.7	23.0
s1423	4895	18	5857	19.6	14	5647	15.3	21.9
s5378	12712	22	14941	17.5	12	14104	10.9	37.7
s9234	19020	4	21011	10.5	2	20108	5.7	45.7
s13207	33733	13	35125	7.1	11	34985	3.7	47.9
s38417	103719	160	124276	19.8	145	119269	15	24.2
s38584	104362	5	112173	7.5	3	110536	5.9	21.3
des_perf	900082	215	959793	6.6	188	948384	5.4	18.2
Average								28.9

of buffers when padding before/after relocating through the proposed method and LP method, and column 6 and 9 show the runtime respectively. The results show that the result of proposed heuristic method is close to the linear programming method but it consumes much less runtime than the LP method.

Finally, Table 3 compares the cost of timing speculation for the following two cases. One is that we directly use logic synthesis tool to meet the short path constraints. The other is that we first apply our techniques to reduce the number suspicious FFs, heuristically pad the short path, and then use the synthesis tool to get the area report. Column 2 indicates the area of the original circuits without any timing speculators inserted. Columns 3 and 6 show the number of suspicious FFs in the circuit before and after applying our technique, from which we can see the number of suspicious FFs is cut down. Column 4 and 7 demonstrate the total area of the circuit after inserting timing speculators without. Columns 5 and 8 show the percentage of the additional area for timing speculation. Finally, column 9 demonstrates the total benefits we can get from our optimization method combining both reduced suspicious FFs and padding. From the result, we can see that the area cost for timing speculation is mainly related to the number of suspicious FFs in the circuit and the circuit size itself, and the proposed retiming and short path padding techniques are able to achieve significant cost reduction for timing speculation.

6. CONCLUSION

In this paper, we propose an optimization technique for cost-efficient circuit-level timing speculation through re-synthesis. The optimization includes two steps. First is to reduce the number of suspicious FFs by relocating FFs and second is to efficiently pad short paths from a global perspective. Experimental results show that the cost for circuit timing speculation is largely cut down with our methods.

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] 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)*, pp. 329–333, 2006.

[2] 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.

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

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

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

[6] P. Franco and E. McCluskey, "Three-Pattern Tests for Delay Faults," in *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 452–456, 1994.

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

[8] M. Favalli and C. Metra, "Sensing circuit for on-line detection of delay faults," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 130–133, 1996.

[9] Y. Tsiatouhas, S. Matakias, A. Arapoyanni, and T. Haniotakis, "A sense amplifier based circuit for concurrent detection of soft and timing errors in CMOS ICs," *Proc. IEEE International On-Line Testing Symposium (IOLTS)*, pp. 12–16, 2003.

[10] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 86–94, 1999.

[11] T. Austin and V. Bertacco, "Deployment of better than worst-case design: solutions and needs," in *Proc. International Conference on Computer Design (ICCD)*, 2005, pp. 550–555.

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

[13] 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)*, pp. 172–179, 2009.

[14] J. Leiserson, C. Saxe, "Retiming synchronous circuitry," *Algorithmatica*, pp. 5–31, 1991.

[15] J. Baumgartner, "Min-area retiming on flexible circuit structures," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 176–182, 2001.

[16] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 398–402, 1993.

[17] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra, "Circuit Failure Prediction and Its Application to Transistor Aging," in *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 277–286, 2007.

[18] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 156 –161, 1993.

[19] B. Taskin and I. Kourtev, "Delay insertion method in clock skew scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 4, pp. 651 – 663, 2006.

[20] S.-H. Huang, C.-H. Cheng, C.-M. Chang, and Y.-T. Nieh, "Clock period minimization with minimum delay insertion," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2007, pp. 970 –975.

[21] M. Chatterjee, S. Banerjee, and D. K. Pradhan, "Buffer assignment algorithms on data driven asics," *IEEE Transations on Computers*, pp. 16–32, 2000.

[22] D. Tong, E. Young, C. Chu, and S. Dechu, "Wire retiming problem with net topology optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1648–1660, 2007.

[23] C. Lin and H. Zhou, "An Efficient Retiming Algorithm Under Setup and Hold Constraints," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2006, pp. 945–950.

[24] J. Roy, S. Adya, D. Papa, and I. Markov, "Min-cut Floorplacement," in *IEEE Trans. on Computer-Aided Design*, vol. 25, no. 7, pp. 1313 – 1326, 2006.