Memristor-Based Clock Design and Optimization with In-situ Tunability

Shuyu Kong, Jie Gu, and Hai Zhou
EECS Department, Northwestern University, Evanston, IL, U.S.A.

***Abstract*—Process variation is the dominating factor for performance degradation in modern IC chips. The conventional guard-band design methodology leads to significant performance penalty. This paper utilizes an emerging non-volatile resistive device, memristor, with timing violation detectors to dynamically achieve local recovery from timing violation during the runtime, eliminating the necessity of testing phase. It develops a systematic self-tuning mechanism that globally adjusts the clock skew scheduling to compensate the timing violation, and determines the tunability of the memristor-based self-tunable circuits. It also proposes an algorithmic memristor placement across the clock tree to balance the tradeoff between hardware cost and system tunability. Experimental results show that our approach can improve the yield from 90% to 98% with only 4% overhead in average.**

I. Introduction

With aggressive scaling down of feature sizes in VLSI fabrication, process variation has become a critical issue on the integrated circuit design. Traditional design achieves the timing yield specification by allocating adequate design margin in the pre-silicon design stage, which restricts the frequency scaling in high performance circuits. To relieve the pessimistic restriction imposed on the pre-silicon design phase, Post-Silicon-Tunable Buffer was proposed [1] and has been widely accepted as a promising solution to adjust the clock skew scheduling in the post-silicon stage, thus mitigating the impact of process variation on circuit performance. Previous research on PST Buffer can be categorized into two directions: (1) improving the PST buffer design to support the variation aware capability [2]; (2) exploring clock tree synthesis optimization, PST buffer placement strategy, and tuning methodology [3]–[5]. However, CMOS based PST not only incurs large area overhead for storage of configuration bit but also requires repeating calibration and tuning whenever chip is powered up. In addition, there was no online detection reported in previous PST approach and thus it does not provide the necessary improvement for timing violation within local circuits.

Recently, the fourth passive circuit element found by Chua [6] has raised significant attention from the IC design community. The unique property of memristor, that it is able to permanently store its resistance value, makes it a perfect candidate device in non-volatile solid-state memory design [7]. Gu and Li [8] explored the memristor application in self-resilient design and demonstrated a novel memristor-based self-tunable circuit that can perform runtime timing violation detection and dynamic tuning. The design borrows the self resilience concept from Razor [9]. However, the Razor approach requires continuous detection and flush of pipeline to remove the impact of errors, and thus does not provide a "removal" method for the variation source compared with the proposed scheme in this paper. On the other hand, [8] is based on experimental analysis and lacks of a systematic way of solving the detection and tuning sequence.

The biggest challenge in self-tunable circuits is how to design a systematic tuning mechanism that can detect infeasible clock periods. The basic idea of self-tuning is to borrow time among different flipflop stages. A clock period is doomed to be infeasible if there exists at least one cycle formed by flipflop stages whose average delay (i.e. the total delay divided by the number of flipflops) is larger than the clock period. The local tuning method [8] cannot detect this situation. Since the timing error detection circuit cannot identify the source flipflop of the timing error, the idea of finding the infeasibility by identifying such a cycle does not work either.

We discover that a systematic self-tuning is similar to a concurrent algorithm for longest path on the circuit timing graph. By exploring the similarity, we developed a self-tuning mechanism that can detect infeasible clock periods. Our mechanism can also be extended to self-tunable circuits where the memristor-based tunable buffers are located in the clock tree. In this situation, a buffer can affect the clock latency of multiple flipflops. We developed the logic circuit to control each tunable buffer and compare the overhead with the original circuits.

We also investigate the memristor placement and sizing problem using the pre-silicon statistical delay information. Here the goal is to minimize the total number of memristor used while maintain a target yield. Our approach is based on Statistical Static Timing Analysis (SSTA) and the results are confirmed using Monte-Carlo simulations.

Our contribution in this paper is as follows:

- To the best of our knowledge, it is the first work to study the systematic tuning strategy for memristor-based circuits with clock tree.
- We propose a heuristic but effective methodology to place tunable buffers with memristors by analyzing FF criticality based on SSTA with redefined statistical operations to improve analysis accuracy.

II. Memristor Based Tunable Circuit

This section reviews the error-detection circuits similar to razor flipflop [9] and the proposed tunable clock buffers as basic units in our work. Figure 2 shows the conceptual view of the proposed memristor-based circuit with clock tree for setup time resilience. The circuit structure of tunable buffer and emergency detector is demonstrated in Figure 1. Each critical FF is connected with an emergency detector. The memristor is implemented in a Veriloga model integrated into Cadence Virtuoso circuit schematics and we assume a tuning resistance between 50 ohm to 100k ohm [8]. One tunable buffer can add up to approximately 100ps delay. Cascading multiple tunable buffers can achieve more tunable delay. The maximum detectable timing violation amount is determined by the length of detection window in the emergency detector. In every clock cycle, the detector will feed the "violation" signal into the tuning control logic generating "Tune" signal. By tuning the resistance value of the memristor in the buffer, a tunable delay can be generated. Ideally, the detecting and tuning will proceed in

2159-3477/17 $31.00 © 2017 IEEE
DOI 10.1109/ISVLSI.2017.81

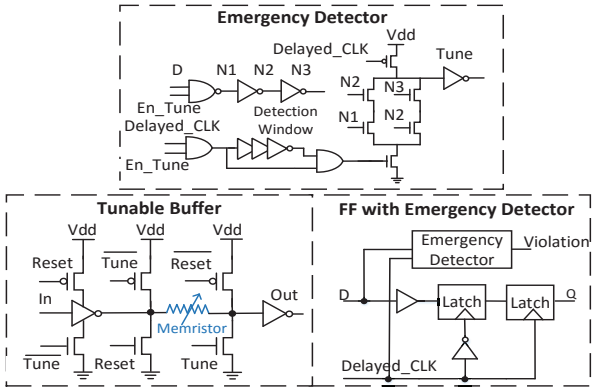

Figure 1. Basic component of memristor-based circuit

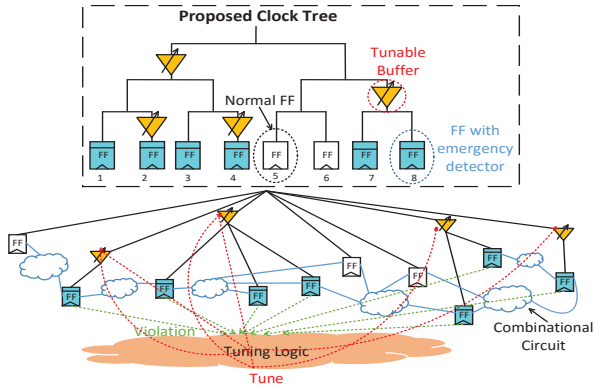

Figure 2. Conceptual memristor-based circuit with clock tree

every cycle until the setup time violations are fully removed. The tuning can be achieved In-Situ and does not need to be applied continuously or repetitively as in the case of previous PST. The proposed buffer leads to significant area saving, tuning energy as well as system operation time compared with previous Razor FF or PST.

Similar sequential circuit for hold-time resilience can be constructed by placing tunable buffer in the data path. We realize that adjusting clock skew scheduling does not help when combinational paths from one flipflop to another simultaneously violate setup and hold time constraint. However, if extra delay is inserted on the short path to avoid hold time violation while not affecting the long path, setup time constraint can be satisfied by increasing the clock latency of the destination flipflop. In this sense, this paper only leverages the setup time healing mechanism and assumes the hold time constraint can always be solved by delay insertion or padding [10].

III. Problem Formulation

The setup time constraint of a sequential circuit is modeled with a set of inequalities as follows.

$$\forall i, j \in V \text{ and } i \rightsquigarrow j, t_i + D_{ij}^{max} < T_{clk} - \theta_j + t_j \quad (1)$$

Where V is the set of all FFs in the sequential circuit. t_i and t_j are the clock skew for FF i and j respectively, θ_j is the setup time for j, $i \rightsquigarrow j$ representing there are combinational paths from i to j, among all of which D_{ij}^{max} is the delay of the longest path. In the memristor-based tunable circuit, the clock skew of every flipflop consists of two components as illustrated below:

$$\forall i \in V, t_i = t_i^s + L_i \quad (2)$$

where t_i^s and L_i are the static and dynamic clock skew for FF i respectively. Since buffers are placed on the clock trees, L_i is equivalent to the total tuning amount of buffers controlling i, explicitly expressed as:

$$\forall i \in V, L_i = \sum_{k \in AM_i} l_k \quad (3)$$

where AM_i consists of all the ancestor buffers for i and the buffer at the same node as i if any. l_k is the tuning amount of buffer k. The memristor in the buffer does not support backward tuning, thus:

$$\forall k \in M, l_k \geq 0 \quad (4)$$

where M is the set of all the buffers placed on the clock tree.

From hardware point of view, setup time constraint is interpreted as 1 bit information, whether there is timing violation or not. The exact timing slack for a FF cannot be measured. Based on the aforementioned, we formulate the main problem as follows.

Problem 1 (Placement Problem)**.** *Given a circuit with clock tree, optimize the memristor-based tunable buffer placement to minimize the total number of memristors while maintaining the target yield.*

Problem 2 (Tuning Problem)**.** *Given circuit with memristor-based tunable buffer placed on the clock tree, utilize emergency detectors and buffers to determine the tunability of the given circuit and adjust the clock skew scheduling under the tuning constraint (2)-(4) to remove setup time violations if the circuit is tunable.*

We first discuss tuning problem in Section IV. Then we present our approach to place the buffers on the clock tree in Section V.

IV. In-Situ Self-Tuning Mechanism

Our goal in this section is to decide a hardware strategy to self-tune the circuits and determine the conditions under which the circuit is considered failing the recovery of timing violation with the given clock period.

A. Basic Tuning

Given a circuit with delay information under a fixed clock period, the clock skew scheduling to eliminate all the timing violations can be obtained mathematically. But in practice, it's unrealistic for a circuit to perform online self-measurement of the precise setup timing violation amount. The information that can be extracted from the violation detector is just 1 bit: either there is timing violation or not. So we aim to implement a hardware tuning algorithm that can dynamically control the tuning and detecting phase so that the tuning result will match the theoretical results as much as possible. For simplicity, we start from the following assumptions:

1) Memristor-based tunable buffers are placed at the leaf node of the clock tree so that every buffer can only affect the clock latency of one flipflop. Let buffer i tunes the FF i so that $\forall i \in V, L_i = l_i$

2) The memristor has a tuning resolution, which is the minimal amount of latency it can tune in one step, defined as d.
3) The allowed initial maximum setup time violation is $d * \delta$, which means any single violation can be tuned correctly in at most δ cycles.
4) The tunable ranges provided by buffers are large enough to not become a limiting factor in the tuning process.

Based on the above assumptions, we present a basic tuning algorithm, Leaf FF Tuning Algorithm.

Algorithm 1 Leaf FF Tuning

```
1: procedure TUNING PROCEDURE
2:     cycle @ 0
3:     ∀i ∈ M, l_i = 0
4:     for cycle @ 1 to |M| * δ do
5:         if ∃i ∈ V, i detects setup time violation then
6:             ∀i ∈ V with violation, tune clk skew of i by d
7:         else
8:             return tuning success
9:     cycle @ |M| * δ + 1
10:    if ∃i ∈ V, i detects setup time violation then
11:        return tuning failure
```

Each iteration in Leaf FF Tuning Algorithm is one clock cycle of tuning. All the violation flipflops are tuned concurrently. The dynamic clock skew l for the flipflops not controlled by buffers are fixed at 0. The circuit is tunable if and only if there exists dynamic clock skew assignments, each of which is integer multiples of d, such that constraints from equations (1)-(4) are all satisfied. The validity of the algorithm is based on Theorem 1.

Theorem 1. *If there exists feasible tuning to eliminate all the setup timing violation , The Leaf FF Tuning Algorithm can find it within $|M| * \delta$ clock cycles, where $|M|$ is the total number of buffers.*

The correctness proof of theorem 1 is based on the intuition that after every δ clock cycles, the tuning amount of at least one more buffer is fixed and needs not to be tuned again if the whole circuit is tunable. Theorem 1 implies if there is still violation after $|M| * \delta$ clock cycles, the violated circuit is not tunable and a tuning failure should be reported. The untunability discovered by our algorithm is either a result of unavoidable overtuning under the given memristor tuning resolution or due to the existence of negative timing slack cycles in the circuits. In our proposed algorithm, we tune the buffer by memristor tuning resolution d. Alternatively, a larger tuning amount λd, $\lambda > 1$, can be selected in every tuning operation to speed up the tuning process, but at the risk of losing tunability.

Figure 3 shows an example to better illustrate our Leaf FF Tuning Algorithm. There are 6 FFs in the circuit and all the combinational paths are represented by arrows with delay values. Here we assume every FF with emergency detector($FF2$, $FF3$, $FF4$, $FF5$) is connected to a memristor-based tunable buffer while the skew of normal FFs ($FF1$, $FF6$) is fixed at 0. Since there are 4 buffers in total and the initial max violation is 2, circuit should be tuned successfully within 8 clock cycles and in this case, only 6 clock cycles are needed. The tuning procedure showing the skew of each FF in every clock cycle is listed. If the delay between $FF5$ and $FF2$ increases by 1, tuning will not finish after 8th clock cycle and tuning failure will be reported. This is consistent with

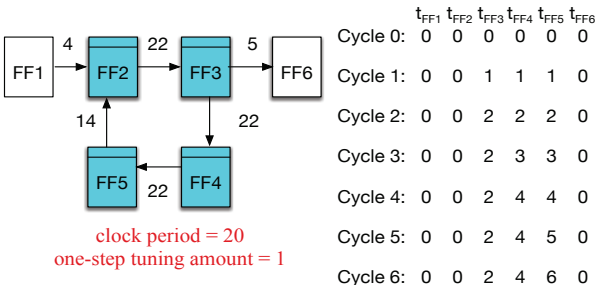

Figure 3. memristor tuning example

the theoretical results since there is a path cycle constructed by $FF2$, $FF3$, $FF4$ and $FF5$ with negative overall timing slack. Intuitively, our tuning algorithm provides a criterion indicating when to stop due to the tuning infeasibility. If we instead apply the simple tuning mechanism described in [8] on circuits with untunable timing violations, we have to keep adjusting the clock skews of violated FFs forever and never realize that it is a waste of effort.

B. Generic Tuning

Leaf Tuning Algorithm only deals with the situation where all the buffers are on the leaves of the clock tree. However, if buffer reduction technique is applied, some buffers can be placed on the non-leaf nodes of the clock tree. We propose a more complex hardware mechanism to tune the circuit with more general buffer placement.

Similar with the previous tuning algorithm, we intend to increase the clock latency by d in every cycle for each FF that has timing violation. However, because there is no one-to-one mapping between FF and buffer, some buffers may affect the clock skews of more than one FFs. In certain violation situations, we can not tune the violating FFs without affecting the non-violating FFs. One straightforward strategy is to tune the greatest common ancestor buffer of all the violating FFs. However, such method is very likely to result in overtuning. An proper tuning mechanism to reach the feasible clock scheduling, if any, should tune violating FFs in every cycle while affecting non-violating FFs as less as possible.

We define a term DC associated with every buffer, where DC_i is the the set of the FFs that either is on the same clock tree node as buffer i, or does not have buffer on the same leaf node but has i as its least ancestor. Every FF in DC is "directly connected" with i, meaning there is no intermediate buffer between them. The tuning signal for every buffer is given as follows.

$$sel_i = \bigvee_{j \in DC_i} violation_j$$

$$tune_i = sel_i \wedge \neg(\bigvee_{j \in AM_i \setminus i} sel_j)$$

$$violation_i = \exists j \in V, L_j + t_j^s + d_{ji} + \theta_i > L_i + t_i^s + T$$

In every cycle, we want to increase the clock arrival time of any FF with timing violation only by a latency of d to avoid overtuning. Therefore, all the ancestor buffers of such FFs are the potential candidates to be tuned. In our methodology, we prioritize to tune the least ancestor of the violating FF. We call such buffer as "selected" buffer. All the "selected" buffers have sel equal to $TRUE$ at current clock cycle. A selected buffer will not be tuned

only if at least one of its ancestor buffers is also "selected". This ensures no FF has clock skew delayed by a latency greater than d in every cycle. The described tuning logic also guarantees to tune the smallest subset of the non-violating FFs. Therefore, if any non-violating FF is tuned, it is an inevitable step to reach the feasible clock skew scheduling.

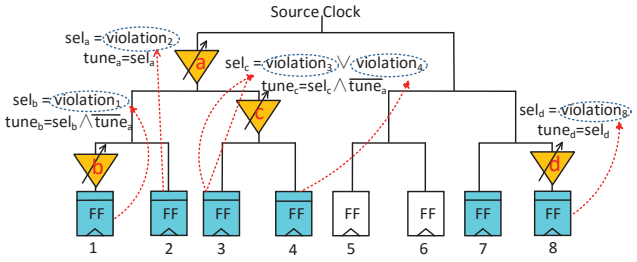

Figure 4. tuning logic for tunable memristors on clock tree

An example of tuning logic design is shown in Figure 4. Note that the $tune$ signals of buffer a and d are not dependent on the $tune$ signal of any other buffers because they do not have ancestor buffers on the clock tree. Among the 4 buffers, only c "directly connects" with more than 1 FFs, which is reflected in the expression of sel_c.

Algorithm 2 Clock Tree Tuning

```
1:  procedure TUNING PROCEDURE
2:      cycle @ 0
3:      ∀i ∈ M, l_i = 0
4:      for cycle @ 1 to |M| * δ do
5:          if ∃i ∈ M, tune_i = TRUE then
6:              ∀i ∈ M with tune_i = TRUE, l_i ← l_i + d:
7:          else
8:              return tuning success
9:      cycle @ |M| * δ + 1
10:     if ∃i ∈ M, tune_i = TRUE then
11:         return tuning failure
```

Again, the correctness of Clock Tree Tuning Algorithm is based on the following theorem.

Theorem 2. *Given buffer placement on clock tree, if there exists feasible tuning to eliminate all the setup timing violation, Clock Tree Tuning Algorithm can find it within $|M| * \delta$ clock cycles, where $|M|$ is the total number of buffers.*

Up to now, we still maintain the assumption that all the buffers on clock tree have enough memristors so that the tuning range is not a restriction in approaching the feasible clock skew scheduling. But if a more general and realistic placement is concerned, we have to take the tunable range at each node of clock tree into consideration. Accordingly, the tuning logic design should be revised because sel signal becomes more complex. If the selected buffer is already tuned to the upper range bound, we have to select its least ancestor where tunable range is not used up. This process can been viewed as the upward propagation of selection. To explain the revised tuning logic, we define a new set DM for every buffer. DM_i is the set of buffers whose least ancestor buffer is i. In addition, we introduce another three signals for every buffer: $SAIR$, $SAOR$ and CBT. $SAIR_i$ is true if and only if buffer i is selected and has not used up its tuning range. $SAOR_i$ is true if and only if i is selected but already tuned to its upper bound r_i so that it will propagate $selection$ upward to its ancestors. CBT_i is $TRUE$ if and only if at least one ancestor buffer of i is determined to be tuned in the current cycle. To describe the CBT, we define $LAM(i)$ as the least ancestor buffer of i. Note if buffer i does not have ancestors, then $CBT_i = FALSE$ by default.

$$SAIR_i = (\bigvee_{j \in DC_i} violation_j \vee \bigvee_{j \in DM_i} SAOR_j) \wedge (l_i < r_i)$$
$$SAOR_i = (\bigvee_{j \in DC_i} violation_j \vee \bigvee_{j \in DM_i} SAOR_j) \wedge (l_i = r_i)$$
$$CBT_i = CBT_{LAM(i)} \vee TUNE_{LAM(i)}$$
$$TUNE_i = SAIR_i \wedge \neg CBT_i$$

Intuitively, $SAIR$ can be treated as same as the sel signal in the previous tuning algorithm where tuning range is not considered.

Similarly, we claim that if the circuits is tunable, there is no timing violation any more after $|M| * \delta$ clock cycles. Applying the similar reasoning we used for the previous two tuning algorithms, we can prove the correctness of this more generic and comprehensive tuning algorithm(we leave all the formal proof out due to the space limitation).

V. Tunable Buffer Placement Optimization

In this section, we propose a methodology to place tunable buffers with memristors based on SSTA. We first place buffers on the leaves of the clock tree according to the FF *timing criticality*. Then, we apply a heuristic technique to reduce total memristor numbers.

A. Placement With SSTA

To overcome the tuning range limit, a tunable buffer may be embedded with multiple memristors to control a self-healing FF. Now the question is how to assign memristors to different buffers that controls different FFs? The proper memristor allocation strategy should take the "timing criticality" and potential timing violation amount of diffrent FFs into consideration, that is, the FF with higher probability to have larger violation amount should be provided with larger tunable range to compensate the negative slack. That requires us to apply statistical timing analysis (SSTA) to evaluate the "timing criticality" of each FF. In SSTA, every combinational path delay is represented as a Gaussian random variable. Our goal is to estimate the distribution of the latest arrival time for every FF.

During the SSTA, we have to repeatedly apply two fundamental operations: max and sum. The sum operation is relatively easy and exact. The max of two Gaussian distributions can be approximated to another Gaussian distribution as in [11] [12]. We deploy Clark Approximation [13] to obtain the first and second moment of the max. During the calculation, we find that most max operations are over a constant and a Gaussian random variable. A constant value can also be represented by a Gaussian distribution with $\sigma = 0$. However, we realize that the results obtained through such approximation are too inaccurate. Therefore, we introduce a tuple to represent timing variable. The tuple, e.g. $\{(\mu, \sigma), C\}$, refers to the max of the Gaussian variable (μ, σ) and a constant C. With this new representation included, we derive the max and sum operations between tuples to replace the orignal operations.

$$\{(\mu_A, \sigma_A), C_A\} + \{(\mu_B, \sigma_B), C_B\} = \{(\mu_s, \sigma_s), C_A + C_B\}$$
$$max(\{(\mu_A, \sigma_A), C_A\}, \{(\mu_B, \sigma_B), C_B\}) = \{(\mu_m, \sigma_m), max(C_A, C_B)\}$$

where (μ_s, σ_s) and (μ_m, σ_m) are derived to be the max of some gaussian distributions shown below, and can be computed with Clark Approximation.

$$(\mu_s, \sigma_s) = max((\mu_A + \mu_B, \sqrt{\sigma_A^2 + \sigma_B^2}), (\mu_A + C_B, \sigma_A), (\mu_B + C_A, \sigma_B))$$
$$(\mu_m, \sigma_m) = max((\mu_A, \sigma_A), (\mu_B, \sigma_B))$$

By applying the redefined operations, we can estimate the latest arrival time for every FF and allocate memristors to provide tuning range accordingly. As illustrated in our memristor placement algorithm, $|V| - 1$ iterations are needed to calculate the arrival time distributions for all FFs. This is because violation amount can accumulate and propagate across multiple stages. One iteration of calculation can only capture the delay information in one combinational path. The worst case is when all FFs are pipelined and the latest arrival time distribution of the last FF may not get stable until $|V|-1$ iterations of the calculation. In practice, far less iterations are needed for the calculation to converge if the clock period is not too aggressive. Heuristically, we run the Monte Carlo simulation for circuits without memristors and find a proper clock period resulting in reasonable yield(e.g. 90%).

Given the stabilized arrival time distribution for all FFs and the target timing yield TY_t, we can calculate the tuning amount needed for each FF according to the following theorem:

Theorem 3. *Assuming a low dependence between the arrival time of all the critical FFs, the yield P every FF should achieve is approximated by $TY_t^{\frac{1}{|V_c|}}$, where $|V_c|$ is the number of critical FF.*

We define the critical FF as those FFs with static clock skew t^s and latest arrival time distribution $\{(\mu, \sigma), C\}$ that satisfy either $t^s < C$ or $t^s < \mu + 3\sigma$. It is reasonable to maintain the assumption in Theorem 3 because under a proper clock period, there are only few critical FFs which are most likely far apart from each other and not sharing common path.

Following the convention, we only consider assigning tunable buffers to critical FFs. The tuning amount the tunable buffer should provide to a critical FF is $max\{C, \mu + \lambda_p \sigma\} - t^s$, where λ_p is the value to achieve yield $P = TY_t^{\frac{1}{|V_c|}}$ in standard normal distribution. This would ensure the individual critical FF to be tunable with a high probability of approximately $TY_t^{\frac{1}{|V_c|}}$. Then, based on Theorem 3, we can achieve the target yield of the whole circuit.

B. Memristor Reduction Method

Algorithm 3 only allocate memristors to the tunable buffers on the leaf node. Advanced memristor reduction technique can be performed to minimize hardware cost. We apply a greedy method which moves memristors upward as much as possible. For example, if N_1 and N_2 memristors are allocated to two buffers at different child nodes of the clock tree and $N_1 > N_2$, then N_2 memristors can be assigned to the buffer at the parent node so that the number of the memristors on both child nodes will decrease by N_2. The overall effect reduces N_2 memristors. The reduction operation can be iteratively applied until no more memristors can be moved up. Such reduction method serves the purpose of minimizing the total memristor number. Meanwhile, we realize our method neglects the clock arrival time correlation between different FFs and may end up deteriorating the overall tunability of the circuit.

Algorithm 3 Placement Algorithm

```
a_i: the clock arrival time for i in a tuple form
a'_i: the clock arrival time for i in next iteration.
r_m: maximum tuning range for a single memristor
n_i: the number of memristors needed to tune FF i
procedure
    Initialize all flipflop arrival time:
    ∀i ∈ V, a_i = {(−∞, 0), t_i^s}
    for count from 1 to |V| − 1 do
        ∀i ∈ V
        a'_i = max(a_i, max_{∀j∈V, j⇝i} (a_j + d_ji − T + t_j^s − t_i^s))
        if ∀i ∈ V, a'_i ≈ a_i then
            stop iteration
    ∀i ∈ V_c estimate n_i based on a_i, |V_c|, r_m and TY_t
```

VI. Experimental Results

A. Implementation Discussion

We implement our tuning and memristor placement algorithm in C++ and conduct experiments on ISCAS89 circuit benchmarks. All experiments are conducted on a 2.7Hz Linux machine with 8GB RAM. Mean delay of each gate is assigned based on the fanout. The delay variance is randomly generated but restricted within 10% of the delay mean. For each benchmark, we run 1000 iterations of Monte Carlo Simulation and apply placement and tuning algorithm on the circuit with the simulated delay information.

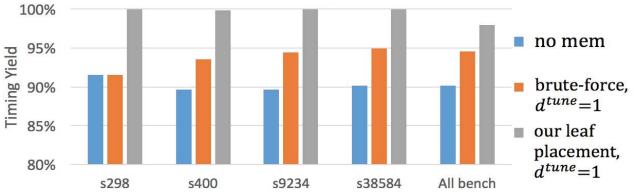

Figure 5. yield comparison: our placement vs brute-force placement.

We implement 3 versions of memristor placement. Two versions are based on our placement algorithm and the only difference is one doesn't apply memristor reduction technique so that all the memristors are on the leaf while the other one does. For comparison, we include a brute force leaf placement which uses the same number of memristors as our strategy without memristor reduction technique and allocate every FF same number of memristors. This brute-force placement is the natural choice for design in [8] where no memristor placement strategy is proposed. We also have three versions of generic tuning algorithm with different normalized one-step tuning amount $d = 1, 50$ and 100 where 1 is the assumed to be the minimum tuning amount for memristor. The max tuning amount for one memristor is assigned as 100.

B. Experimental Result Analysis

Figure 5 illustrates the benefit of memristor tuning, which in average achieves 8% yield improvement over conventional design with non tuning mechanism. The observation that the yield improvement with our memristor placement methodology in general doubles the improvement with brute force placement shows our

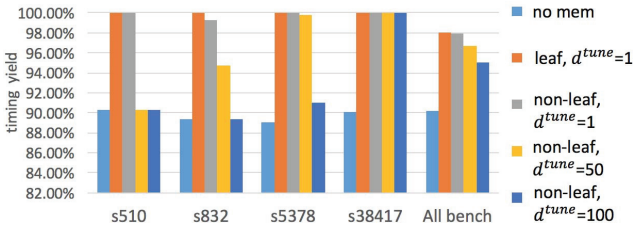

Figure 6. yields for different placements and tuning strategies.

	Overhead %		Num of Memristors	
circuit	leaf	non-leaf	leaf	non-leaf
s713	5.43	5.94	12	10
s1423	8.47	7.59	48	40
s13207	2.72	2.94	215	162
s35932	19.1	16.10	7946	4705
s38417	8.88	10.16	4196	2666
All bench	3.97	4.26	571.4	358.4

Table I. Overhead of implementing tuning logic circuit and number of memristors for different placements given original yield=90%

strategy is much more capable of capturing the FF criticality information. Combining Figure 6 with Table I, we find when memristors are inserted on non-leaf nodes of the clock tree, total number of memristors is significantly reduced while losing trivial amount of timing yield, which further validates the advantage of our memristor placement and reduction algorithm.

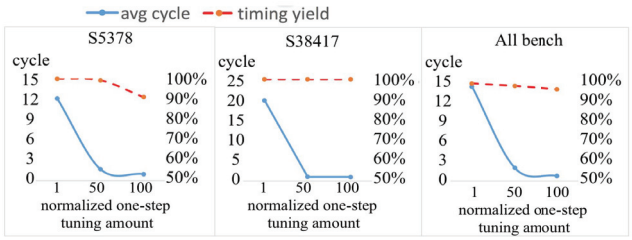

Figure 7. average tuning cycles along with timing yield for different one-step tuning amount

To study the trade-off between tuning cost and timing yield, we obtain the average cycles for all cases that successfully tune the setup time violations with different one-step tuning amount. As shown in Figure 6, if one-step tuning amount d^{tune} is lower, the tuning resolution is higher and it is less likely to overtune the circuit, thus achieving a higher timing yield. On the other hand, Figure 7 shows that as one-step tuning amount increases, the tuning cycles decreases much faster than the timing yield in most cases. We conclude that tuning the minimum resolution is generally too conservative to be the optimal solution and a relatively larger one-step tuning amount with no overtuning will dramatically speed up the tuning process and meanwhile maintain high timing yield.

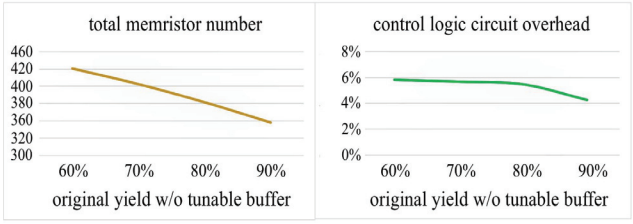

Figure 8. impact of clock period on the placement overhead in average of 30 ISCAS89 circuit benchmarks

Control logic overhead is estimated based on basic gate counting. We realize that the overhead in practice can be higher due to global routing. According to Figure 8, the overhead is impacted by the clock period. If the yield without tuning is low, meaning the clock period is aggressive, then more memristors and control logic gates are needed to maintain a target yield after tuning. In addition, Table I shows that overhead varies between different circuit benchmarks. Overhead is below 10% for most of the benchmarks. However, s35932 has overhead close to 20%, which is due to relatively high portion of critical paths in the circuits.

VII. Conclusion and Acknowledgement

In this paper, we present a memristor tuning and allocating methodology for a proposed self-healing circuit design. Our tuning algorithm provides circuits with the intelligence of judging the overall tunability. Our memristor placement algorithm efficiently finds the critical FFs and reduces the number of memristors on the clock tree while maintaining the timing yield as shown in the experimental results.

This work is partially supported by NSF under CNS-1441695 and CCF-1533656, and by SRC under 2014-TS-2559.

References

[1] S. Rusu and S. Tam, "Clock generation and distribution for the first ia-64 microprocessor," *JSSC*, 2000.
[2] V. B. Suresh and W. P. Burleson, "Variation aware design of post-silicon tunable clock buffer," *ISVLSI*, 2014.
[3] J. Tsai, D. Baik, C. C. Chen, and K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling," *ICCAD*, 2004.
[4] J. Tsai, Z. Lizheng, and C. C. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," *ICCAD*, 2005.
[5] M. Kaneko, "Timing-test scheduling for constraint-graph based post-silicon skew tuning," *ICCD*, 2012.
[6] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507–519, 1971.
[7] R. Williams, "How we found the missing memristor," *IEEE Spectrum*, vol. 45, no. 12, 2008.
[8] J. Gu and J. Li, "Exploration of self-healing circuits for timing resilient design using emerging memristor devices," *ISCAS*, 2015.
[9] S. Das, C. Tokunaga, S. Pant, W. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "Razorii: In situ error detection and correction for pvt and ser tolerance," *IEEE Journal of Solid-state Circuits*, vol. 44, no. 1, pp. 32–48, Jan 2009.
[10] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Minimum padding to satisfy short path constraints," in *ICCAD*, 1993.
[11] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," in *IEEE TCAD*, 2006.
[12] H. Chang and S. Sapatnekar, "Statisitical timing analysis considering spatial correlations using a single PERT-like traversal," in *ICCAD*, San Jose, CA, Nov. 2003, pp. 621–625.
[13] C. E. Clark, "The Greatest of a Finite Set of Random Variables," *Operations Research*, vol. 9, no. 2, pp. 145–162, 1961.