Design and Synthesis of Self-Healing Memristive Circuits for Timing Resilient Processor Design

Shuyu Kong, Hai Zhou, *Senior Member, IEEE*, and Jie Gu, *Member, IEEE*

***Abstract*— Modern microprocessors suffer from significant on-chip variation at the advanced technology nodes. The development of CMOS-compatible memristive devices has brought nonvolatile capability into silicon technology. This paper explores new applications for memristive devices to resolve performance degradations that result from process variation. Novel self-healing flip-flop and clock buffers are developed to automatically detect timing violation and to perform timing recovery by tuning the resistance values of memristor devices. To incorporate the circuit techniques into VLSI circuits design, novel device placement and tuning algorithms have been developed. The proposed design methodology is demonstrated in a 45-nm fast Fourier transform processor design. Our test results show that performance gains of up to 20% can be achieved using the proposed self-healing circuits, with only 1% area**

***Index Terms*— Memristor, process variation, self-healing, sequential circuits, timing resilient.**

I. Introduction

As CMOS devices are scaling down into the 10-nm regime and beyond, extremely small feature size has introduced major difficulty in controlling the power and performance of transistors, leading to significant process variations [1], [2]. For planar transistors, e.g., 22 nm and above, the major phenomena that contribute to the variability of transistors include random dopant fluctuation (RDF), oxide thickness variation, and line-edge roughness (LER), and so on [3], [4]. Fig. 1(a) shows a threshold voltage variation through the generations of CMOS technology [5]. As can be seen in Fig. 1(a), the variability continuously increases through the technology nodes. Such an exacerbated variability has posed major challenges in the design and manufacturing of modern microprocessors. Fig. 1(b) shows the amount of power required to cope with the threshold voltage variation in order to maintain a constant speed target, based on the simple power calculation $P = \alpha CV_{dd}^2$. For more recent FinFET technology, the RDF has been reported to be significantly reduced due to the use of an intrinsic silicon channel [6]. However, several other effects, e.g., metal gate granularity and LER, start to dominate the random process variation [7], [8]. The variability remains as a major challenge to the technology.

Manuscript received September 2, 2017; revised January 12, 2018 and March 10, 2018; accepted April 25, 2018. This work was supported in part by NSF under Grant CNS-1441695 and Grant CCF-1533656 and in part by SRC under Grant 2014-TS-2559. *(Corresponding author: Shuyu Kong.)*

The authors are with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 USA (e-mail: shuyukong2020@u.northwestern.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2834827

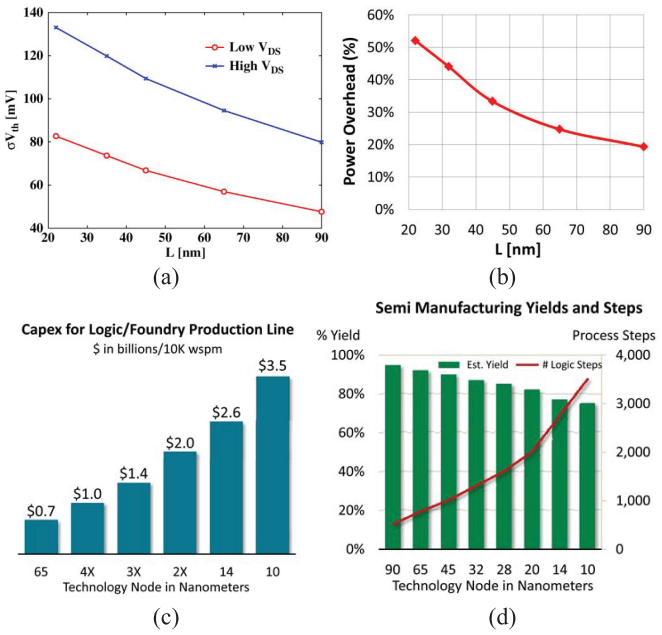


Fig. 1. (a) Threshold voltage variation in recent generations of CMOS technology [5]. (b) Associated power overhead due to process variation that is required to maintain the target speed of the devices. (c) Foundry spending on different technology nodes. (d) Yield from different technology nodes [14].

Alongside variability, several other effects, such as aging and supply noise variation, also introduce significant performance uncertainty in modern CMOS chips. The most common mechanisms of aging-induced reliability degradation include negative bias temperature instability (NBTI), hot carrier injection (HCI), and time-dependent dielectric breakdown (TDDB) [10]–[13]. NBTI and HCI introduce a threshold voltage increase, while TDDB can lead to more catastrophic functionality failure. To deal with the aging-induced threshold voltage increase, a constant design margin (5%∼10%) has been traditionally allocated to account for the degradation over the lifetime of the microprocessor. Unlike the static process variation, reliability degradation happens over time and thus cannot be detected during production tests. It must thus be either compensated in real time or budged with a most conservative estimation.

Besides the ever-rising performance variation of silicon chips, manufacturing costs also escalate in each generation of the CMOS technology. Fig. 1(c) and (d) shows the manufacturing costs and yield loss in the 90–10-nm scale. Lower yield of chip fabrication is converted into a further increase of the developing cost and a waste of natural resources [14].

All of the above-mentioned technology-induced uncertainty points to a critical missing feature of existing silicon technology, i.e., a "self-healing" technique that allows the hardware to detect and recover from its defective performance degradation. To enable such a "self-healing" capability, three basic capabilities must be present in a technology: 1) a detection capability which allows defect discovery; 2) a recovery capability which offers a repair of defects; and 3) a nonvolatile memory capability which allows the repair to be memorized and permanently deployed. Conventional CMOS technology has been partially equipped with the first two capabilities, e.g., detection and recovery, while the third, nonvolatile memory, has not been realized. One example of existing detection and recovery operation is the well-known Razor technique, where the performance degradation is detected and recovered by a flush operation in the pipeline [15], [16]. Although the Razor technique has introduced an intelligent self-detection and correction mechanism, there are significant limitations on its usage. First, the error detection and correction can only fix a sparse timing violation event, for instance, that results from a sudden supply drop. If the emergent condition continues to exist, for example, due to the aging-induced delay increase, there is no correction that can improve the situation. Second, the technique is more suitable for a CPU design with flush and replay capability for error correction, which may not be presented for general application-specific integrated circuits designs. Third, as the error detection and correction action must be constantly ON, significant power is consumed during the process of detection. In addition, all the error correction effort is lost when the power is OFF, and hence, there is no "healing" of existing problems. Standard CMOS technology is thus still missing an important nonvolatile memory feature that could be used to permanently "heal" a defective chip.

Fortunately, recent development in CMOS-compatible nonvolatile memory devices, e.g., memristor or resistive RAM (RRAM), provides a strong extension to the existing chip technology. Currently, most memristor applications reside in the design of memory and neuromorphic circuits where the reconfigurability and nonvolatility of the device lead to significant power saving and processing enhancement [17], [18]. In addition, several nonmemory or nonneuromorphic usages of memristors are being proposed. For instance, Rose *et al.* [19] proposed a new reconfigurable logic family using memristors. Cong and Xiao [20] proposed to use a fully integrated memristor to build a field-programmable gate array (FPGA) to reduce the storage overhead of the conventional FPGA. It has been shown that an area saving of five times can be achieved to replace the SRAM with the memristor in an FPGA design. Shin *et al.* [21] proposed to build a programmable gain amplifier for analog application. Göknar *et al.* [22] and Werner and Gregory [23] proposed a modulator design based on the frequency response of the memristor device. Laiho *et al.* [24] proposed an analog arithmetic computing unit to extend the potential application of the device.

Leveraging the unique features of a memristive device, i.e., tunable resistance and permanent memory of its tuning state, this paper explores a fundamentally new application, a "self-healing" capability, toward the goal of healing timing violations introduced by the random process variation, which is currently difficult to be globally compensated through existing voltage or clock scaling, e.g., dynamic voltage and frequency scaling. Our contributions in this paper include: 1) the design and implementation of novel healing circuits, including self-healing flip-flop (FF) circuits and self-healing clock buffers using memristive devices to provide the required detection, recovery, and memory features for self-healing functionality; 2) the development of a complete synthesis flow and placement method to implement the circuit techniques in a realistic processor design; and 3) the development of a generic CAD algorithm that optimizes the placement and tuning process of the memristive device, leading to a robust and efficient healing operation. Our proposed techniques are demonstrated in a 45-nm design with the use of memristors fully integrated into the traditional design flow.

The novelty of this paper is: 1) to the best of our knowledge, this is the first work that leverages RRAM technology to perform self-healing against design variation in modern chips; 2) this is also the first work to analyze and discuss the online global tunability in the circuit and provide a theoretical threshold beyond, which the circuit is considered "doomed"; and 3) we present a novel heuristic placement algorithm based on the statistical static timing analysis (SSTA) to efficiently allocate tunable-buffers on the clock tree.

II. Model of Memristor and PCELL Design

The so-called fourth fundamental element, i.e., "memristor," was first introduced from HP Lab in 2009 [25], [26]. The development of the memristor device completed the search for a missing element, which relates magnetic flux with the charge through a conductor. Mathematically, it can be expressed as

$$d\varphi = M(\rho)dq \tag{1}$$

$$V(t) = M(\rho)I(t) \tag{2}$$

where φ is the flux or the time integral of voltage, q is the charge or the time integral of current, and M is the memresistance. As a result, unlike other passive components, such as resistor, capacitor, or inductor, memristor shows a continuous hysteresis effect where the present state of the device depends on the operating history of the device. The hysteresis can be used as a memory, which stores the information represented by the current state (resistance) of the device. Unlike binary memory devices, such as MRAM and RRAM, memristor can have continuous states and thus can be continuously tuned to a desirable resistance value. Fig. 2 shows the drawing of the two-terminal device configuration using titanium dioxide films as the active layer and platinum as the contact. The resistance of a memristor is determined from the simplified equations

$$M(\rho) = (1-\rho)\cdot R_L + \rho \cdot R_H \tag{3}$$

$$\frac{d\rho(t)}{dt} = \mu_v \cdot \frac{V(t)}{M(\rho)} \cdot \frac{R_L}{h^2} \tag{4}$$

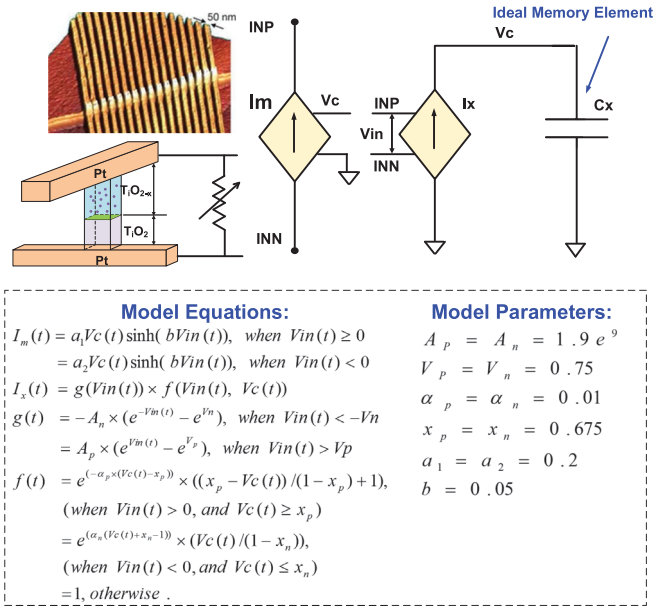


Fig. 2. Basic memristor device structure and spice model used in this paper.

where $M(\rho)$ is the memristance, p is the relative doping front positions between 0 and 1, μ_v is the equivalent mobility of dopants, and h is the thickness of the thin film. When the voltage across the memristor is beyond its threshold, the doping front position will shift changing the resistive value of the device. By applying pulses to the device, the device can be tuned with a resistance between R_H and R_L, with a large tunable range from a few hundreds of ohms to a few hundreds of kilo-ohms or megaohms.

To be able to correctly quantify the transient behavior of our device within a CMOS circuit, we performed the following tasks. First of all, the VerilogA model was derived based on the previous publication from the HP Lab that has been verified with measurement [27]. The VerilogA model can be simulated in Cadence Virtuoso to quantify the resistance change under the influence of voltages. The detailed spice model, model equations, and used parameters in this paper are provided in Fig. 2. Note that an ideal capacitor C_x is used to provide the hysteresis behavior of the device. The size of the active junction region of our memristor is 50 nm $\times$ 50 nm. Fig. 3 shows simulated switching waveforms of the model used in this paper, validating the dynamic behavior of our model that aligns with what has been shown in the previous publication [27].

We also implemented the parametric cell (PCELL) in a 45-nm CMOS technology to perform a layout exercise of our design. Because a memristor can be inserted between any metal layers in the backend process [28], [29], we placed our memristor between metal 3 and metal 4 in the design, as shown in the layout example in Fig. 8. The developed PCELL allows the users to choose metal layers for memristor insertion and the X–Y dimension of the memristor. As can be seen from the layout, the memristor and supporting transistors can be made extremely compact due to the fact that the memristor stays on the top of the active layer. With the supporting PCELL and the spice model, we can simulate the real-time behavior of the

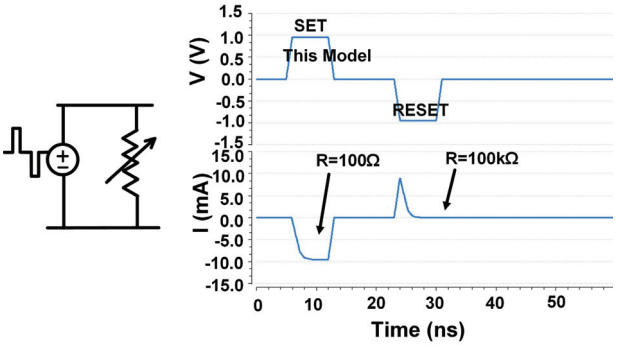


Fig. 3. Simulated transient behavior of our spice model.

proposed memristor-based "self-healing" circuits as discussed in Section III.

III. Self-Healing Flip-Flop Design

The self-healing feature requires a fine-tuning capability in the circuits. Although discrete time steps can be programmed into a timing critical circuit, such programmability incurs large overhead and is volatile with power recycling. Instead, we propose to use the "resistive" nature of the memristor to fine-tune the delay of timing sensitive circuits and use the "memory" nature of the memristor to store the tuning state effectively removing the overhead of programmability. In this section, we discuss the proposed self-healing FF, which features timing detection and timing recovery using memristor devices. The proposed FFs are able to automatically detect a timing "emergency" condition where the data and clock edges fall into the predefined hazardous window. Upon detection, the device will tune itself by increasing data path delay or clock delay to avoid a timing failure to happen. It is worth highlighting the difference between the proposed design and the existing Razor FFs. The Razor FFs could detect a timing emergency but rely on a system pipeline reissue to recover the failure. In addition, the detection has to be constantly enabled. In contrast, in the proposed design, the tuning operation is only activated for a short period to allow the circuit to perform detection and recovery. After the tuning operation, the circuit has been "healed," and no more detection is needed, thus saving the power overhead of the detection operation. In the proposed design, the timing failure is permanently removed as compared with the detection only operation from the Razor technique. The tuning operation can be issued at the chip startup or reissued by the system under a significant operating condition change, e.g., temperature.

Fig. 4 shows the design of the self-healing FF for hold resilience. A tunable memristor is deployed at the input of the data path. The memristor is set into its lowest resistance at the startup to avoid additional delay. In the tuning mode which is activated by the "En_Tune" signal, when a timing emergency condition is detected, a pulse derived from the clock signal is provided to continuously increase the delay of the data path until no more timing emergency exists. The detection of hazardous timing condition is realized by a dynamic transition detector, as shown in Fig. 4. The Tune_clk derived from

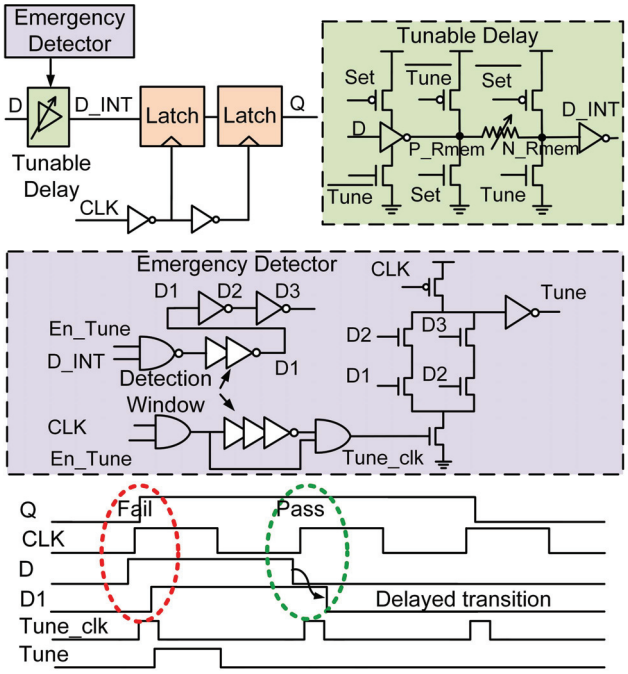


Fig. 4. Proposed hold-healing FF.

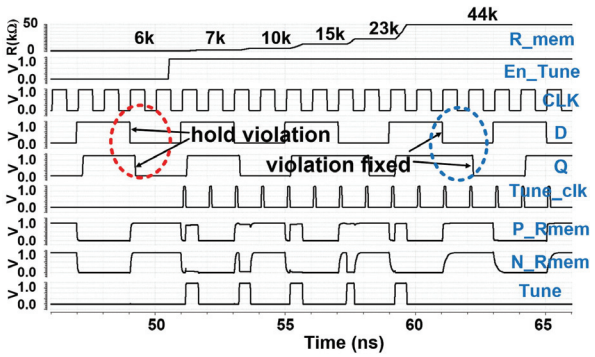


Fig. 5. Simulated waveforms of the hold-healing operation.

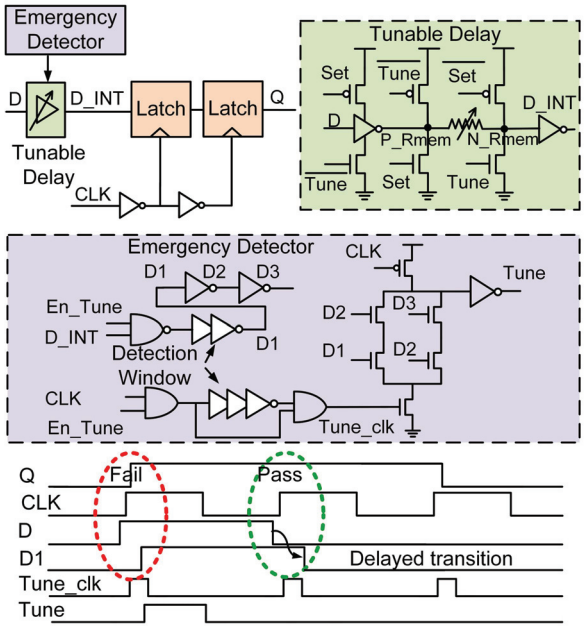


Fig. 6. Proposed setup-healing FF.

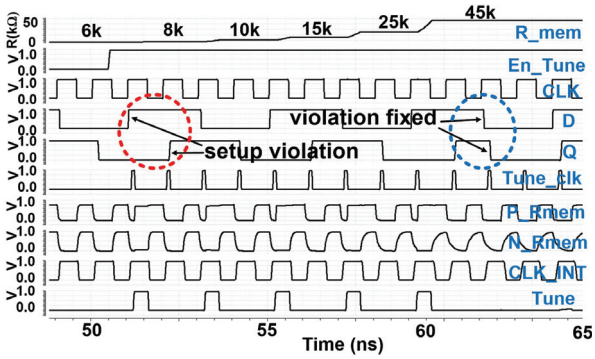


Fig. 7. Simulated waveforms of the setup-healing operation.

the clock signal provides a detection window. If D1–D3, a delayed version of the input data, toggle within the predefined window, the timing is considered not reliable, and thus, a "tune" pulse is issued to inject current into the memristor. As a result, the increased resistance of the memristor slows down the data path and fixes a hold violation. The detection window is generated by the delay of an inverter chain and determines the maximum amount of timing violation that this circuit can detect. Fig. 5 shows a simulated waveform with a 75-ps hold violation. The simulation was performed at 0.95 V, −40 C, and slow corner in a 45-nm CMOS technology. At the end of the operation, the hold violation is removed by tuning the memristor from 6 to 44 kΩ. Despite that a polarity change at D input (N_Rmem or P_Rmem) is generated to tune the memristor, there is no impact to the internal data value of the FF, as the data input is gated by the first latch when the clock is high. Essentially, by automatically tuning the resistance on the data buffer inside the FFs, we can permanently "heal" the hold timing violation.

Figs. 6 and 7 show the similar design of the self-healing FF for setup resilience and its operating waveforms under an 85-ps setup violation. The tunable device is placed in the clock buffer instead of the data buffer as in the hold fixing FF. At the end of tuning, the setup violation has been removed with a memristor resistance increased to 45 kΩ. Overall, our simulation shows that for hold healing, a maximum of 110-ps tunable delay can be produced from the tuning circuit. Considering the variation of detection circuit itself based on Monte Carlo simulation, a worst case of 70-ps hold violation can be recovered. For setup FF, a maximum of 140-ps delay compensation can be produced. Considering the variation of the detection circuit, a worst case of 100-ps setup violation can be recovered. Fig. 8(a) shows the layout comparison of the proposed self-healing setup FF with the standard cell design FF. Embedded memristor PCELL is also pointed out in the layout. Due to the inclusion of the emergency detector, the area has doubled from the conventional design. The area impact to the overall design will be assessed in Section VII.

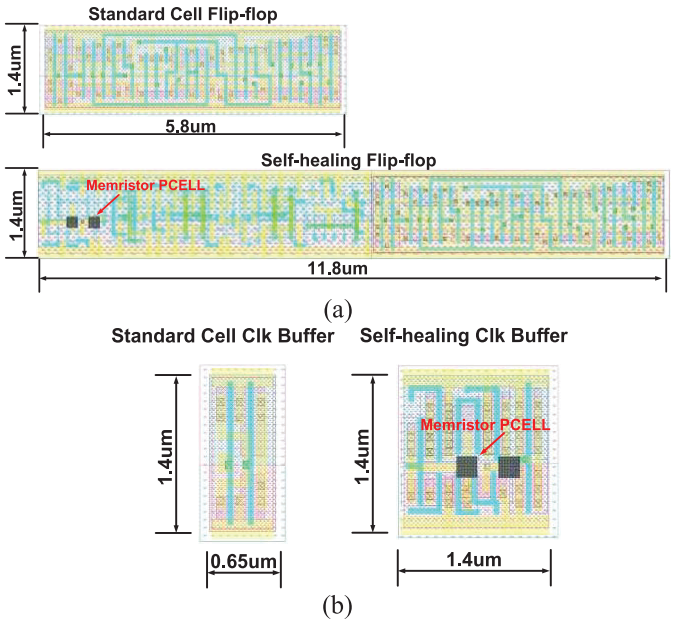


Fig. 8. (a) Layout comparison between conventional FF and self-healing FF (setup). (b) Layout comparison between conventional clock buffer and self-healing clock buffer.

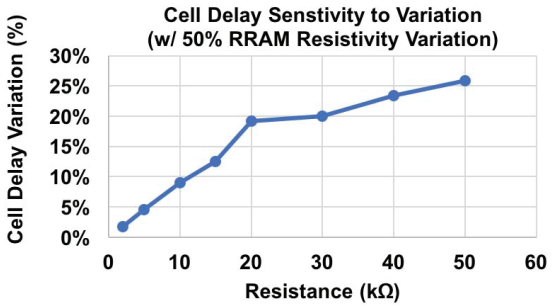


Fig. 9. Cell delay sensitivity to variation with respect to the resistance of the memristor.

Fig. 9 demonstrates the sensitivity of circuit delay to drift of resistance by qualifying the delay increase in term of ps versus the percentage of resistance change. It shows that at low resistance state, the sensitivity is very small, because the delay is more dominated by transistors rather than the memristor. But at high resistance state, this variation increases. At system level, we can consider this drift overtime and reissue a tuning process based on this sensitivity information.

Overall, we believe our scheme is much less susceptible to process variation compared with other applications, such as analog neuromporphic computing. This is because we have a feedback-based tuning setting and we perform incremental tuning. Our tuning algorithm discussed later in Section VII makes a reasonable assumption on the tuning steps and changes at each cycle based on our device model presented in Section II (characterized with measurement result in the literature and simulated in spice as shown in Fig. 5). We also set a threshold for the maximum tuning cycles, beyond which we decide that the circuit is untunable. That threshold is applicable as long as the memristance will increase, and there is a guaranteed minimum resistance increase in every

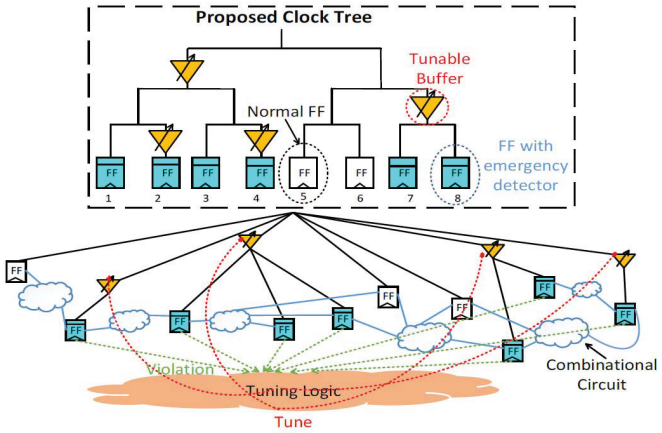


Fig. 10. Self-healing clock buffers used in a clock tree.

tuning cycle. In addition, tuning can be performed several times to set back the resistance if drift happens over a certain amount time. Therefore, the RRAM variability will not significantly impact our generic algorithm.

IV. Self-Healing Clock Buffer Design

In previous discussions, the tunable cell is implemented together with the corresponding FF to achieve timing resilience. This is essentially similar as placing buffers on the leaf node of the clock tree. To obtain further optimization, cells with tuning capability can also be implemented across clock trees, that is, tunable buffers can be allocated on the nonleaf nodes of the clock tree. Compared with the self-healing FF design, implementing tunable timing cells across clock tree has the following benefits: 1) the tunable clock cells can be shared by placing the cells into higher levels of the tree leading to a saving of area; 2) tunable clock cells do not reside on the critical timing paths of the design and hence have less impact on the timing closure of the design; and 3) more tuning range can be achieved by cascading multiple clock cells.

To explore the capability of the self-healing clock buffer design, we use the tunable clock buffers, as shown in Fig. 6. The emergency detector is located in the leaf FF and is similar as what has been used in the FF design in Section III. In addition, each tunable clock buffer consists of two regular inverters with tunable memristor sandwiched in between to create tunable delay. The tuning enable signal is issued from the emergency detector at the FF along with a central controller. One tunable buffer can provide approximately up to 100-ps delay. Cascading multiple tunable buffers can achieve more tunable delay. Fig. 10 shows the conceptual view of the self-healing clock tree design. In every clock cycle, the detector will feed the "violation" signal into the tuning control logic, which generates the "tune" signal. By tuning the resistance value of the memristor in the buffer, a tunable delay can be generated. The detection and tuning will proceed in every cycle until the setup time violations are fully removed.

V. Synthesis of Self-Healing Circuits

As the tuning operation of the circuits effectively changes the timing of the entire processor, a sophisticated design

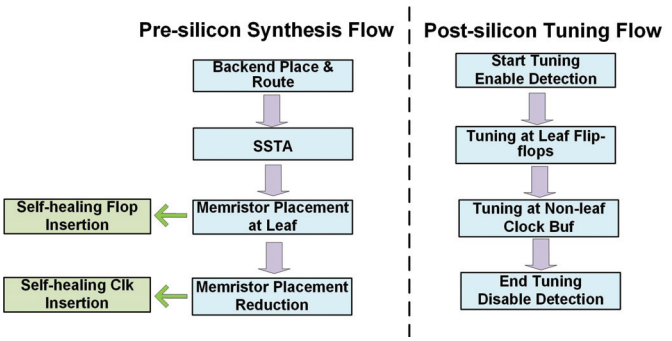


Fig. 11. Design flow of the proposed self-healing circuits.

flow has to be adopted to cope with the stochastic nature of the design. Fig. 11 shows the overview of our proposed design flow to synthesize the use of self-healing FFs and self-healing clocks. During presilicon design phase, the SSTA is used to assess the variability of the design and help optimize the placement of self-healing circuits. A novel algorithm, which will be discussed in Section VI, has been developed to place the tunable FF or clock buffers at optimal locations of the design. During postsilicon operation, self-tuning can be launched during the operation of the chip. A special tuning algorithm, which will be discussed in Section VII, is used to rebalance the timing among stages of the pipeline of the processor design. In the tuning algorithm, the tuning decision is based on the timing violation situation and placement of the self-healing buffer. In other words, the memristor placement in the presilicon phase will decide the tunability of the circuit in the postsilicon online tuning process. It is worth highlighting that, compared with the conventional flow, which pessimistically assigns a design margin to handle the worst case unpredictable variation, our proposed design flow removes the design margins. After the healing process, the memristor-powered self-healing circuits compensate the random process variation and allow the processor to operate at a more optimized condition with balanced delay among pipeline stages. As a result, significant performance gain can be achieved by removing pessimism from the conventional design methodology.

VI. Problem Formulation for Placement and Tuning of Self-Healing Circuits

Without a proper tuning mechanism, FF-based self-healing can only guarantee recovery of local timing violations. The basic idea of self-tuning is to borrow time among different FF stages. However, if there exists at least one cycle formed by FF stages whose average delay (i.e., the total delay divided by the number of FFs) is larger than the target clock period, the whole circuit is doomed to be untunable, and the given clock period is regarded as infeasible. However, the local self-healing cannot detect the infeasible clock period but keeps recovering every reported timing violation. This requires us to develop a strong tuning mechanism that dynamically adjusts the clock skew scheduling and decides the feasibility of the clock period as accurate as possible.

Moreover, FF-based self-healing only considers allocating memristor-based tunable buffers on the leaf nodes of the clock tree. But if we allocate some buffers on the nonleaf nodes, the new placement may achieve better performance or smaller area overhead. Meanwhile, it is important to consider the circuit tunability when assigning memristors and allocating buffers in order to maintain high yield. All these factors should be put into analysis to obtain an optimal placement. Note that in some special situations, it is difficult to fix timing violations. For instance, when combinational paths from one FF to another simultaneously violate setup and hold time constraint, both setup time and hold time self-healing trigger but contradict with each other. However, if extra delay is inserted on the short path to avoid hold time violation while not affecting the long path, setup time self-healing can remove the setup time violation by increasing the clock latency of the destination FF. In this sense, we only leverage the setup time healing mechanism and assumes that the hold time constraint can always be solved by delay insertion or padding.

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

$$\forall i, j \in V \quad \text{and} \quad i \rightarrow j, \; t_i + D_{ij}^{\max} < T_{\text{clk}} - \theta_j + t_j \tag{5}$$

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, and θ_j is the setup time for j, $i \rightarrow j$ representing that 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 FF consists of two components as illustrated in the following:

$$\forall i \in V, \quad t_i = t_i^s + L_i \tag{6}$$

where t_i^s and L_i are the static and dynamic clock skews for FFi, 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

$$\forall i \in V, \quad L_i = \sum_{k \in \text{AM}_i} l_k \tag{7}$$

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

$$\forall k \in M, \quad l_k \geq 0 \tag{8}$$

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

Based on the above, 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 the 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 (6)–(8) to remove setup time violations if the circuit is tunable.

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
```

We first discuss the tuning problem in Section VII. Then, we present our approach to place the self-healing circuits in Section VIII. We also evaluate our algorithms for both the problems in Section IX.

VII. 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 timing graph and a fixed clock period, it is not hard to calculate the clock skew scheduling that satisfies setup time. But in practice, circuits are unable 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. Therefore, 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. Intuitively speaking, our tuning mechanism should fully leverage the tunability of the self-healing circuit. For simplicity, we start from the following basic cases.

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 FF. Let buffer i tune the FF i so that $\forall i \in V$, $L_i = l_i$.
2) There is a minimal amount of latency memristor that can tune in one step, defined as d.
3) The allowed initial maximum setup time violation is within $d * \delta$, which must be smaller than the detection window of the emergency detector. Thus, 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-mentioned assumptions, we present a basic tuning algorithm, leaf FF tuning algorithm (see Algorithm 1).

Each iteration in a leaf FF tuning algorithm is one clock cycle of tuning. All the violation FFs are tuned concurrently. The dynamic clock skew l for the FFs not controlled by buffers is fixed at 0. The circuit is tunable if and only if there exist dynamic clock skew assignments, all of which are integer multiples of d, such that constraints from (5)–(8) are satisfied. The validity of the algorithm is based on Theorem 1.

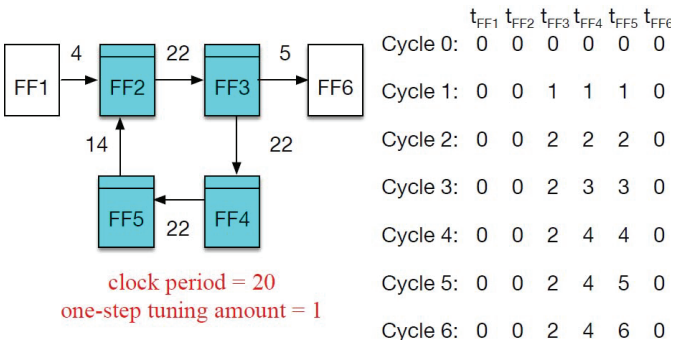


Fig. 12. Memristor tuning example.

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 cycle, 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 that 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 minimum tuning amount or due to the existence of negative timing slack cycles in the circuits.

Fig. 12 shows an example to better illustrate our leaf FF tuning algorithm. There are six FFs in the circuit, and all the combinational paths are represented by arrows with delay values. Here, we assume that every FF with an emergency detector (FF2, FF3, FF4, and FF5) is connected to a memristor-based tunable buffer while the skew of normal FFs (FF1 and FF6) is fixed at 0. Since there are four buffers in total and the initial max violation is 2, the circuit should be tuned successfully within eight clock cycles, and in this case, only six 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 eighth clock cycle and tuning failure will be reported. This is consistent with 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.

B. Generic Tuning

Leaf FF tuning algorithm only deals with the situation where self-healing is inside the FF, meaning that all the buffers are on the leaves of the clock tree. However, if a buffer reduction technique is applied, some buffers can be placed on the nonleaf nodes of the clock tree. We propose a more complex hardware mechanism to tune the circuit with more

general buffer placement. This tuning mechanism covers not only the case where self-healing buffers with memristors are embedded in the FFs, but also the scenario where buffers are allocated outside the FF on the clock tree.

Similar to the leaf FF 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 cannot tune the violating FFs without affecting the nonviolating FFs. One straightforward strategy is to tune the greatest common ancestor buffer of all the violating FFs. However, such a method is very likely to result in overtuning. A proper tuning mechanism to reach the feasible clock scheduling, if any, should tune violating FFs in every cycle while affecting nonviolating FFs as less as possible.

We define a term DC associated with every buffer, where DC_i is 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 that there is no intermediate buffer between them. The tuning signal for every buffer is given as follows:

$$\mathrm{sel}_i = \bigvee_{j \in \mathrm{DC}_i} \mathrm{violation}_j$$

$$\mathrm{tune}_i = \mathrm{sel}_i \wedge \neg \left(\bigvee_{j \in \mathrm{AM}_i \setminus i} \mathrm{sel}_j \right)$$

$$\mathrm{violation}_i = \exists j \in V, \quad L_j + t_j^s + d_{ji} + \theta_i > L_i + t_i^s + T.$$

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 a buffer as a "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 that 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 nonviolating FFs. Therefore, if any nonviolating FF is tuned, it is an inevitable step to reach the feasible clock skew scheduling.

An example of tuning logic design is shown in Fig. 13. 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 four buffers, only c "directly connects" with more than 1 FFs, which is reflected in the expression of sel_c.

Again, the correctness of the clock tree tuning algorithm (see Algorithm 2) 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.

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
```

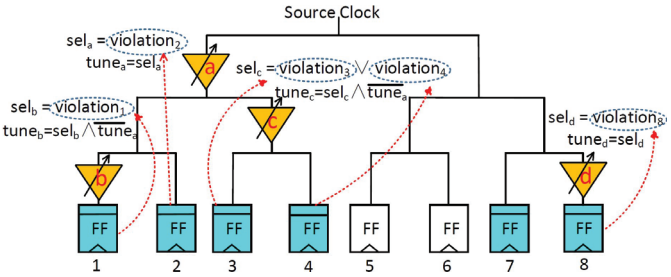


Fig. 13. Tuning logic for tunable memristors on clock tree.

Up to now, we maintain the assumption that all the buffers on the 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 the tunable range is not used up. This process can be 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 $\mathrm{LAM}(i)$ as the least ancestor buffer of i. Note that if buffer i does not have ancestors, then CBT_i = FALSE by default

$$\mathrm{SAIR}_i = \left(\bigvee_{j \in \mathrm{DC}_i} \mathrm{violation}_j \vee \bigvee_{j \in \mathrm{DM}_i} \mathrm{SAOR}_j \right) \wedge (l_i < r_i)$$

$$\mathrm{SAOR}_i = \left(\bigvee_{j \in \mathrm{DC}_i} \mathrm{violation}_j \vee \bigvee_{j \in \mathrm{DM}_i} \mathrm{SAOR}_j \right) \wedge (l_i = r_i)$$

$$\mathrm{CBT}_i = \mathrm{CBT}_{\mathrm{LAM}(i)} \vee \mathrm{TUNE}_{\mathrm{LAM}(i)}$$

$$\mathrm{TUNE}_i = \mathrm{SAIR}_i \wedge \neg \mathrm{CBT}_i.$$

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

Similarly, we claim that if the circuits are 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).

C. Nonideal Tuning Effect

In the previous discussions, we are applying a constant tuning amount d in every cycle. That requires the memristor resistance to increase linearly to the tuning time. However, due to process variation, it is almost impossible to achieve such ideal tuning. In the real circuit model, memristor resistance may increase superlinearly to the tuning time. In other words, the tuning amount becomes larger as tuning proceeds. Such trend increases the chance of overtuning. On the other hand, it can accelerate the tuning process and results in a shorter overall self-healing time.

Instead of being treated as a constant, d is now regarded as the initial tuning amount, which is also a minimum tuning amount during the whole tuning process in a real circuit model as discussed earlier. Besides, d is a controllable parameter and can be used to adjust the balance between the tuning speed and tunablity. Specifically, a larger initial tuning amount d can be used to speed up the tuning process but at the risk of losing tunability.

VIII. 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 number of memristors.

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 different FFs into consideration, that is, the FF with higher probability to have larger violation amount should be provided with the larger tunable range to compensate the negative slack. This requires us to apply 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 [30] and [31]. We deploy Clark approximation [9] 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 original 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 in the following and can be computed with the Clark approximation:

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

By applying the redefined operations, we can estimate the latest arrival time for every FF and allocate memristors to provide the 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 the $|V| - 1$ 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 that every FF should achieve is approximated by $\mathrm{TY}_t^{(1/|V_c|)}$, where $|V_c|$ is the number of critical FFs.

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 the common path.

Following the convention, we only consider assigning tunable buffers to critical FFs. The tuning amount that 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 $P = \mathrm{TY}_t^{(1/|V_c|)}$ in standard normal distribution. This would ensure the individual

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:
    $\forall i \in V$, $a_i = \{(-\infty, 0), t_i^s\}$
    for count from 1 to $|V|-1$ **do**
        $\forall i \in V$, $a_i' = max(a_i, \max_{\forall j \in V, j \hat{} i} (a_j + d_{ji} - T + t_j^s - t_i^s))$
        if $\forall i \in V, a_i' \approx a_i$ **then**
            stop iteration
    $\forall i \in V_c$ estimate n_i based on a_i , $|V_c|$, r_m and TY_t

critical FF to be tunable with a high probability of approximately $TY_t^{(1/|V_c|)}$. Then, based on Theorem 3, we can achieve the target yield of the whole circuit.

B. Memristor Reduction Method

Algorithm 3 only allocates 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 a reduction method serves the purpose of minimizing the total memristor number. Meanwhile, we realize that our method neglects the clock arrival time correlation between different FFs, and may end up deteriorating the overall tunability of the circuit.

IX. Tuning and Placement Evaluation

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

We implement three versions of memristor placement. Two versions are based on our placement algorithm, and the only difference is one that does not apply the 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 the memristor reduction technique, and allocate every FF the same number of memristors.

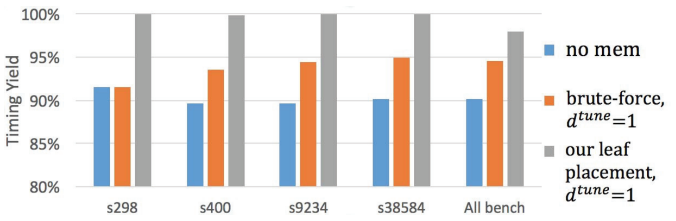


Fig. 14. Yield comparison: our placement versus brute-force placement.

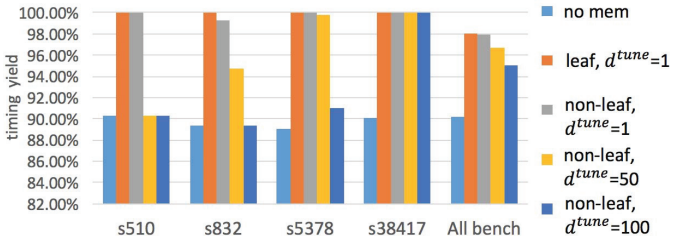


Fig. 15. Yields for different placements and tuning strategies.

TABLE I
Overhead of Implementing Tuning Logic Circuit and Number of Memristors for Different Placements Given Original Yield = 90%

	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

We also have three versions of generic tuning algorithm with different normalized one-step tuning amount $d = 1, 50$, and 100, where 1 is assumed to be the minimum tuning amount for the memristor. The max tuning amount for one memristor is assigned as 100.

Fig. 14 shows the benefit of memristor tuning, which in average achieves 8% yield improvement over the conventional design with nontuning mechanism. The observation that the yield improvement with our memristor placement methodology in general doubles the improvement with brute-force placement shows that our strategy is much more capable of capturing the FF criticality information. Combining Fig. 15 with Table I, we find that when memristors are inserted on nonleaf nodes of the clock tree, the 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.

To study the tradeoff 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 Fig. 16, if one-step tuning amount d^{tune} is lower, it is less likely to overtune the circuit, thus achieving a higher timing yield. On the other hand,

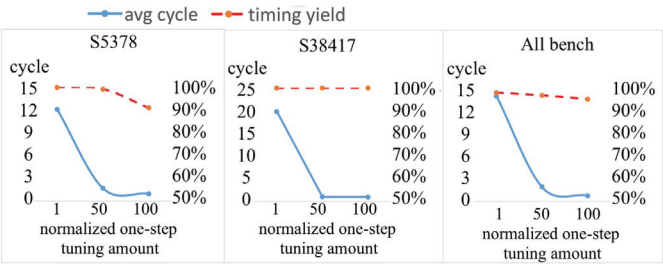


Fig. 16. Average tuning cycles along with timing yield for different one-step tuning amount.

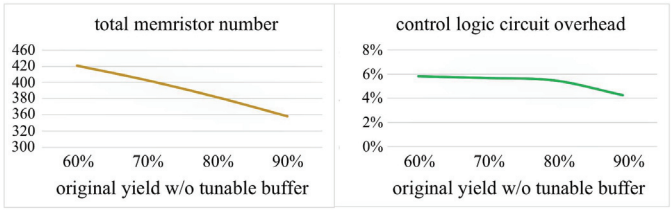


Fig. 17. Impact of clock period on the placement overhead in average of 30 ISCAS89 circuit benchmarks.

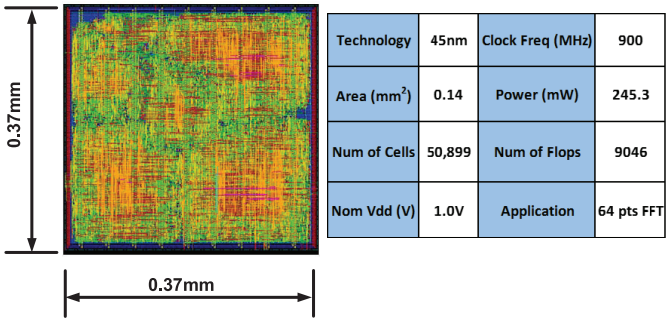
Fig. 16 shows that as one-step tuning amount increases, the tuning cycle decreases much faster than the timing yield in most cases. We conclude that a relatively larger one-step tuning amount with no overtuning will dramatically speed up the tuning process and meanwhile maintain high timing yield.

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 Fig. 17, the overhead is impacted by the clock period. If the yield without tuning is low, meaning that 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 high portion of critical paths in the circuits.

X. Demonstration of Processor Design Using Self-Healing Circuits

To verify the proposed scheme, a 64-point 8-bit highly pipelined fast Fourier transform (FFT) processor was implemented using commercial synthesis and backend tools in a 45-nm technology with a nominal voltage of 1.1 V. Synthesis and place and route were carried out at 0.95 V, −40 C, and slow corner. Fig. 18 shows the FFT processor's layout and design specifications. The FFT processor is synthesized at 1 GHz and operates up to 900 MHz due to on-chip random process variations.

We first demonstrate the operation of the proposed self-healing FFs by performing spice level simulation on extracted critical paths from the FFT processor. Top 50 critical paths from both setup and hold timing closures were physically extracted from Encounter and imported back into Cadence Spectre for Monte Carlo simulation. Each critical path has a self-healing FF converted. Figs. 19 and 20 show the



Technology	45nm	Clock Freq (MHz)	900
Area (mm^2)	0.14	Power (mW)	245.3
Num of Cells	50,899	Num of Flops	9046
Nom Vdd (V)	1.0V	Application	64 pts FFT

Fig. 18. FFT processor layout and design specifications.

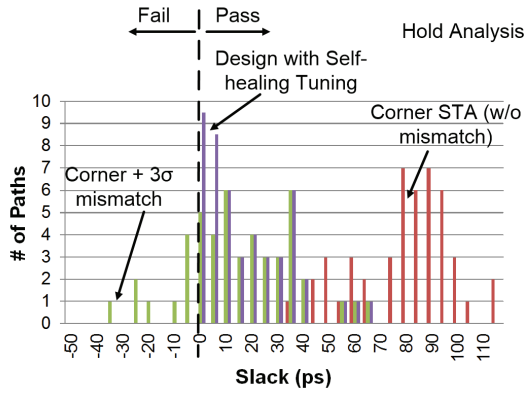


Fig. 19. Hold violation distribution before and after healing.

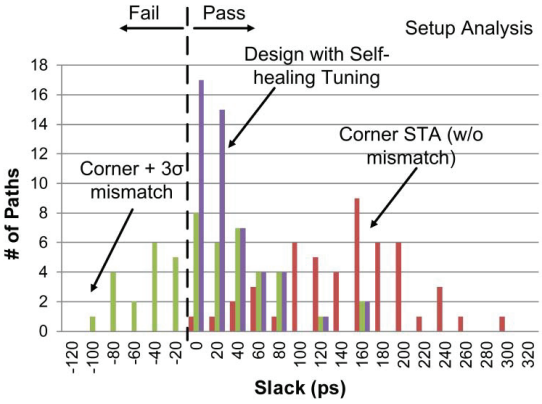


Fig. 20. Setup violation distribution before and after healing.

distribution of the critical paths for setup and hold analysis before and after the self-healing process. The timing histogram shows that the conventional corner-based static timing analysis does not capture the random process violation and hence optimistically estimates the performance of the critical paths. As a result, a slack of −40 to −100 ps is observed in spice level Monte Carlo simulation. For each violated case, the proposed self-healing FF can effectively recover the timing violation upon detection of the timing emergency. As a result, all timing violations have been removed leading to a 10% speed up of the performance.

The above-mentioned test validates the functionality and operation of the proposed self-healing circuits. To further maximize the performance gain, we applied the placement

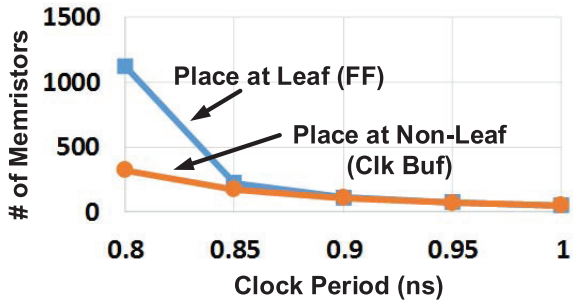


Fig. 21. Numbers of memristor-based circuits placed into the FFT processor based on the target performance (clock period).

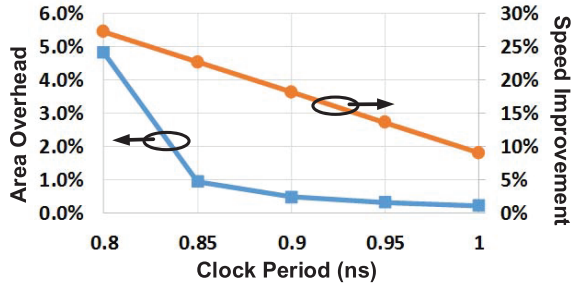


Fig. 22. Area overhead and performance improvement using the proposed self-healing design in the FFT processor.

algorithm in Section VI into the FFT processor. Fig. 21 shows the number of memristor-based circuits to be deployed with different performance targets, i.e., clock period. Both the basic leaf placement, which puts the tuning circuit at the leaf FF, and the memristor reduction technique, which pushes the tuning circuits into higher level of clock trees, are evaluated. As clock speed is set more aggressively beyond 0.85 ns, the number of required memristor devices dramatically increases so as to reach the healing limitation. However, the nonleaf placement outperforms the leaf placement with 3.5 times less number of memristor devices at a maximum frequency target due to the efficient usage of high-level clock buffers.

Fig. 22 shows the overhead of the proposed self-healing design and performance improvement compared with the conventional design. With the proposed self-healing design, the performance has been significantly improved up to 25%. The area overhead remains as small as only 1% when the performance is up to 20%, and increases significantly to 5% when performance is pushed to its limit.

XI. Conclusion

Modern microprocessors face significant timing closure challenges due to on-chip process variation. When dealing with the exaggerated timing issues in advanced technologies, conventional use of design margin significantly trade offs the performance. The recent development of emerging nonvolatile memristive devices provides a new capability in solving the current design dilemma, i.e., balance between performance and design margin. This paper proposes the designs of self-healing FF and clock buffers using memristors to automatically recover the loss of performance due to process variations. New design flow and algorithms are proposed to integrate the proposed circuits to VLSI circuits. A pipelined FFT processor in a 45-nm technology was implemented as a demonstration of the proposed circuits and design methodology. Results shows that the proposed self-healing method can bring 20% performance improvement with only 1% overhead.

References

[1] J. Warnock, "Circuit and PD challenges at the 14 nm technology node," in *Proc. Int. Symp. Phys. Design*, 2013, pp. 66–67.
[2] S. Saxena *et al.*, "Variation in transistor performance and leakage in nanometer-scale technologies," *IEEE Trans. Electron Devices*, vol. 55, no. 1, pp. 131–144, Jan. 2008.
[3] Y. Ye, F. Liu, M. Chen, S. Nassif, and Y. Cao, "Statistical modeling and simulation of threshold variation under random dopant fluctuations and line-edge roughness," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 987–996, Jun. 2011.
[4] G. Leung and C. O. Chui, "Variability impact of random dopant fluctuation on nanoscale junctionless FinFETs," *IEEE Electron Device Lett.*, vol. 33, no. 6, pp. 767–769, Jun. 2012.
[5] G. Panagopoulos and K. Roy, "A physics-based three-dimensional analytical model for RFD-induced threshold voltage variation," *IEEE Trans. Electron Devices*, vol. 58, no. 2, pp. 392–403, Feb. 2011.
[6] A. Loke, "The journey to FinFETs," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2015.
[7] X. Jiang *et al.*, "A device-level characterization approach to quantify the impacts of different random variation sources in FinFET technology," *IEEE Electron Device Lett.*, vol. 37, no. 8, pp. 962–965, Aug. 2016.
[8] S. Markov, A. S. M. Zain, B. Cheng, and A. Asenov, "Statistical variability in scaled generations of n-channel UTB-FD-SOI MOSFETs under the influence of RDF, LER, OTF and MGG," in *Proc. IEEE Int. SOI Conf.*, Oct. 2012, pp. 1–2.
[9] C. E. Clark, "The greatest of a finite set of random variables," *Oper. Res.*, vol. 9, no. 2, pp. 145–162, 1961.
[10] W. Wang *et al.*, "The impact of NBTI on the performance of combinational and sequential circuits," in *Proc. Design Autom. Conf.*, Jun. 2007, pp. 364–369.
[11] A. Kerber, M. Rohner, T. Pompl, R. Duschl, and M. Kerber, "Lifetime prediction for CMOS devices with ultra thin gate oxides based on progressive breakdown," in *Proc. IEEE Int. Rel. Phys. Symp.*, Apr. 2007, pp. 217–220.
[12] J. B. Velamala, V. Reddy, R. Zheng, S. Krishnan, and Y. Cao, "On the bias dependence of time exponent in NBTI and CHC effects," in *Proc. IEEE Int. Rel. Phys. Symp.*, May 2010, pp. 650–654.
[13] R. Arora and J. D. Cressler, "Operating voltage constraints in 45-nm SOI nMOSFETs and cascode cores," *IEEE Trans. Electron Devices*, vol. 60, no. 1, pp. 132–139, Jan. 2013.
[14] (2013). *Entegris Analyst Report*. [Online]. Available: http://investor.entegris.com/node/15166/html
[15] S. Das *et al.*, "RazorII: *In situ* error detection and correction for PVT and SER tolerance," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 32–48, Jan. 2009.
[16] K. A. Bowman *et al.*, "A 45 nm resilient microprocesor core for dynamic variation tolerance," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 194–208, Jan. 2011.
[17] D. Niu, Y. Chen, and Y. Xie, "Low-power dual-element memristor based memory design," in *Proc. Int. Symp. Low Power Electron. Design*, 2010, pp. 25–30.
[18] M. Soltiz, D. Kudithipudi, C. Merkel, G. S. Rose, and R. E. Pino, "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1597–1606, Aug. 2013.
[19] G. S. Rose, J. Rajendran, H. Manem, R. Karri, and R. E. Pino, "Leveraging memristive systems in the construction of digital logic circuits," *Proc. IEEE*, vol. 100, no. 6, pp. 2033–2049, Jun. 2012.
[20] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *Proc. IEEE Int. Symp. Nanosci. Architecutre (NANOARCH)*, Jun. 2011, pp. 1–8.
[21] S. Shin, K. Kim, and S.-M. Kang, "Memristor applications for programmable analog ICs," *IEEE Trans. Nanotechnol.*, vol. 10, no. 2, pp. 266–274, Mar. 2011.
[22] I. C. Göknar, F. Öncül, and E. Minayi, "New memristor applications: AM, ASK, FSK, and BPSK modulators," *IEEE Antennas Propag. Mag.*, vol. 55, no. 2, pp. 304–313, Apr. 2013.

[23] D. H. Werner and M. D. Gregory, "The memristor in reconfigurable radio frequency devices," in *Proc. IEEE Antenna Propag. Soc. Int. Symp. (APSURSI)*, Jul. 2012, pp. 1–2.
[24] M. Laiho, E. Lehtonen, and W. Lu, "Memristive analog arithmetic within cellular arrays," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, 2012, pp. 2665–2668.
[25] D. B. Strukov, G. S. Snider, D. R. Stewart, and S. R. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, Jun. 2008.
[26] R. S. Williams, "How we found the missing memristor," *IEEE Spectrum*, vol. 45, no. 12, pp. 28–35, Dec. 2008.
[27] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor SPICE model and crossbar simulation based on devices with nanosecond switching time," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Aug. 2013, pp. 1–7.
[28] S. Dubosi. *Crossbar Resistive RAM (RRAM): The Future Technology for Data Storage*. Accessed: Dec. 12, 2017. [Online]. Available: http://www.snia.org/sites/default/orig/DSI2014/presentations/HotTopics/SylvainDuBoise_Future_Technology_final.pdf
[29] Crossbar. *ReRAM Advantages*. Accessed: Dec. 12, 2017. [Online]. Available: https://www.crossbar-inc.com/en/technology/reram-advantages/
[30] C. Visweswariah *et al.*, "First-order incremental block-based statistical timing analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2170–2180, Oct. 2006.
[31] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, 2003, pp. 621–625.

Shuyu Kong received the B.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2015. He is currently working toward the Ph.D. degree at the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA.

His current research interests include statistical timing analysis, formal verification, security information flow analysis, and hardware logic encryptions.

Hai Zhou (SM'04) received the B.S. and M.S. degrees in computer science and technology from Tsinghua University, Beijing, China, and the Ph.D. degree in computer sciences from The University of Texas at Austin, Austin, TX, USA.

He is currently an Associate Professor of Electrical Engineering and Computer Science at the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA. His current research interests include security, very large-scale integration computer-aided design, algorithm design, and formal methods.

Dr. Zhou was a recipient of the CAREER Award from the National Science Foundation in 2003.

Jie Gu (M'10) received the B.S. degree from Tsinghua University, Beijing, China, the M.S. degree from Texas A&M University, College Station, TX, USA, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA.

He was an IC Design Engineer at Texas Instruments, Dallas, TX, USA from 2008 to 2010, where he was involved in ultralow-voltage mobile processor design and integrated power management techniques. From 2011 to 2014, he was a Senior Staff Engineer at Maxlinear, Inc., Carlsbad, CA, USA, where he was involved in low-power mixed-signal broadband SoC design. He is currently an Assistant Professor at Northwestern University, Evanston, IL, USA. His current research interests include ultralow power mixed-signal VLSI circuit design, hardware and software co-design with integrated power clock management, and emerging device/technology integration.

Dr. Gu has served as the Program Committee and Conference Co-Chair for numerous low-power design conference and journals, such as the International Symposium on Low Power Electronics and Design, the Design Automation Conference, the International Conference on Computer Aided Design, and the International Conference on Computer Design.