

# Improving Transient Error Tolerance of Digital VLSI Circuits Using ROBustness COmpiler (ROCO)

Chong Zhao, Sujit Dey

Department of Electrical Engineering, University of California at San Diego  
9500 Gilman Drive, Mail Code 0409, La Jolla, CA 92093  
{chong, dey}@ece.ucsd.edu

## Abstract

Due to aggressive technology scaling, VLSI circuits are becoming increasingly susceptible to transient errors caused by single-event-upsets (SEUs). In this paper, we introduce two circuit-level techniques to efficiently yet economically improve SEU tolerance of static CMOS digital circuits. We also developed a “ROBustness COmpiler (ROCO)” to integrate these techniques into the existing design flow to achieve high level of reliability at low design cost. Experiment results show that the proposed methodology is able to greatly improve the circuits’ SEU tolerance with zero timing overhead and very limited area penalty.

## 1. Introduction

Due to smaller feature size, lower supply voltage and higher clock frequency, transient errors caused by high-energy neutron or alpha particle strikes, known as “single-event upset” (SEU), are becoming a great threat to the reliability of VLSI circuits. The transient errors can be characterized into two categories. The first one is caused by direct strikes on the sensitive regions of memory devices (such as SRAM cells or flip-flops). The error rate is independent of the clock frequency. The other one is originated by a particle strike on the sensitive region of a combinatorial logic gate, which may induce a transient electrical pulse (modeled as a glitch with certain amplitude and duration). However, this glitch will not become observable soft errors unless it reaches memory elements with enough strength during a certain timing window. This type of soft error is referred to as “single-event transient (SET)” and its error rate increases with the frequency.

Circuit hardening techniques have been adopted to enhance circuit reliability in mission critical applications[1]. However, the associated penalties are unacceptable for main stream applications: excessive use of redundancies will cause high timing and area overhead; the analysis and implementation of the protections require great engineering efforts, leading to elongated design cycle. Therefore, there is not only a need for cost-effective SEU-tolerance enhancement techniques, but also a requirement of integrating these techniques with the existing design flow.

Many research works focused on cost-effective approaches to improve circuit SEU tolerance. Among them, some have designed hardening flip-flop (FF) cells [2] [3]. However, an FF is usually the endpoint of multiple

timing paths so inserting redundancies to the FF lengthens all timing paths. Consider the circuit in Figure 1, if P2 is a critical path with little timing slack, hardening is not allowed at DFF3 because the extra delay will cause timing violation on P2. If solely relying on FF hardening, we have no choice but leaving all gates unprotected. However, since there is large timing slack in path P1, we can improve the error tolerance of P1 by inserting redundancies along the path as long as the extra delay does not exceed the available positive slack; furthermore, P2 might also get certain protection by proper choices of the combinational gates on the path without increasing the delay. This means that hardening FFs is not always feasible or optimized solution.

SET analysis and mitigation techniques targeting combinational logics have also been developed [4][5][6][7]. For example, [7] proposed that transient glitches occurring at different circuit nodes have different chances to become observable errors and only a small number of nodes are highly vulnerable. Although it pointed out that high tolerance could be achieved with limited cost by focusing on these vulnerable nodes, no explicit solution was given. In [4], a cost function of error tolerance was defined based on path delay and then optimized by gate resizing. However, it did not consider all factors during the optimization. Many factors affect the circuit vulnerability collectively, so reducing the effect of one might worsen the effect of other(s). [6] proposed using device with different threshold voltage ( $V_{th}$ ) or varying supply voltage ( $V_{dd}$ ). Although the timing and power penalty might be low, such approaches will cause more unreliability problems and even higher cost during physical design and chip manufacturing.

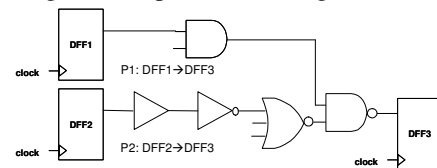


Figure 1 The limitation of hardening DFF

In this work, we introduce two circuit-level techniques, “gate cloning” and “cell resizing”, to enhance the SET tolerance of combinational logics in static CMOS digital circuits. Both techniques have been adopted in the timing closure process [8][9]. With different optimization criterion, they can be used to improve SET tolerance as well. However, due to design constraints, they can only be applied to limited circuit locations. Therefore, we need to

first evaluate the vulnerability of all circuit nodes and select the most vulnerable nodes as our targets. Our vulnerability analysis is based on the “soft spot analysis” proposed in [7]. In order to maximize the efficiency, we develop the “RObustness COmpiler (ROCO)”, an integrated optimization engine that seamlessly interfaces with the current design flow. We will also argue that this can be viewed as part of a “robustness closure” effort.

The rest of the paper is organized as follows: section 2 reviews the soft spot analysis; section 3 and 4 elaborate on gate cloning and cell resizing; section 5 describes the ROCO framework and introduces the concept of robustness closure; section 6 presents the experimental results; section 7 concludes the paper.

## 2. Soft Spot Analysis – A Brief Review

The soft spot analysis [7] is a technique that evaluates the circuit vulnerability based on the circuits’ structure. A “softness” ( $S_N$ ) is calculated for each circuit node  $N$  to measure its vulnerability by considering three masking effects (timing, electrical and logic) that tend to prevent transient at node  $N$  from causing observable errors.

**The timing masking effects ( $T_N$ ):** Circuit primary outputs (POs) are usually registered by D-type flip-flops (DFFs), which only capture data during a sampling window. This poses a timing requirement for noise at an internal node to be captured by a FF because of the propagation delay. This requirement determines the timing factor  $T_N$  and is expressed as a time interval “noise sensitive window”. It means that only if noise at node  $N$  falls into this window, will it be able to arrive at any PO within its sampling window to become a stable error.

**The logic masking effects ( $L_N$ ):** Noise on an internal node can only reach a PO through a sensitized logic path. The logic factor  $L_N$  is measured by the likelihood for noise at node  $N$  to logically reach the POs and is calculated as the probabilities for all side inputs at all gates along the path carrying non-controlling values.

**The electrical masking effects ( $E_N$ ):** Noise must have enough magnitude and duration to propagate through a gate, as depicted in the “noise rejection curve” (NRC) in Figure 2. The x- and y-axis are the width and height of the input noise to a gate, the curve is drawn such that only noise above curve (the “sensitive region”) can propagate through the gate. The NRCs of a gate depends on its size and load: the lower capacitance it drives, the larger the sensitive region (Figure 2(a)); for gates of the same type, the higher the driving strength (for example, INVX2 has a higher driving strength than INVX1), the larger the sensitive region (Figure 2(b)). The electrical factor  $E_N$  is calculated as the ratio ( $R_e^N$ ) of the area of the sensitive region to the area of the “immune region” (region under the curve), and denoted as:  $E_N = R_e^N(\text{gate}, \text{load})$ .

The softness of an individual node  $N$  can be calculated as a product of the three factors:

$$S_N(L_N, T_N, E_N) = T_N * L_N * E_N \quad (1)$$

and the overall circuit vulnerability can be measured by a weighted sum over softness of all circuit nodes:

$$S_{total} = \sum_N w_N * S_N(L_N, T_N, E_N) \quad (2)$$

where  $w_N$  is a designer-specified weighting factor used to emphasize the significance of node  $N$ .

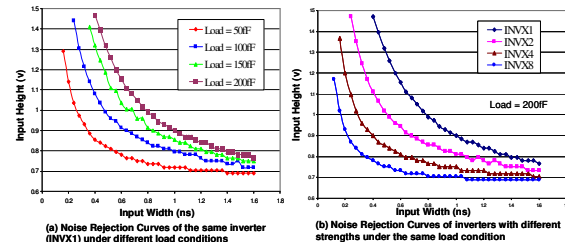


Figure 2 Noise Rejection Curves

Reducing the softness of a node can be realized by reducing one or more of the three factors ( $T_N$ ,  $L_N$  and  $E_N$ ). Next, we introduce two techniques for softness reduction by fine-tuning the circuitry in the vicinity of the soft spots (circuit nodes with high softness values). Both techniques are highly localized so we may assume they will not change the global floor planning, placement and routing.

## 3. Gate Cloning

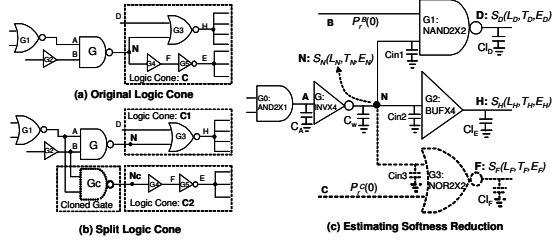
In the circuit segment shown in Figure 3(a), node  $N$  (output of gate  $G$ ) drives a large logic cone  $C$  so a glitch at  $N$  can potentially reach many POs during a large timing window. If we split  $C$  into two smaller logic cones  $C1$  and  $C2$  (Figure 3(b)), respectively driven by  $G$  and a newly created gate  $G_c$  that shares the same inputs as  $G$ , the new circuit is functionally equivalent to the original one but noise at  $N$  or  $N_c$  (the output of  $G_c$ ) will have lower probability of causing functional errors at POs. We call this operation “gate cloning” because the new gate is identical to the original gate and shares part of its functionality. Through gate cloning, a highly vulnerable circuit node is replaced by two (or more) less vulnerable nodes, causing the softness to be redistributed in the vicinity of the changed circuit. Gate cloning not only affects all three factors in softness of the involved circuit nodes but also change the delay of the affected paths.

First, the path delay is affected in two ways. On one hand, gate delays of  $G$  and  $G_c$  in Figure 3(b) are both shorter than the delay of  $G$  in Figure 3(a) because of driving less load. Denoting the delay of a gate  $G$  driving a total load of  $C$  (including the net capacitance and the pin capacitance of the fanout gates) as  $t_G|_{\text{load}=C}$ , the total delay decrease can be calculated as:

$$\delta t_G = t_G|_{\text{load}=C} - \max\{t_G|_{\text{load}=C_1}, t_{G_c}|_{\text{load}=C_2}\} \quad (3)$$

On the other hand, delays of driving gate  $G1$  and  $G2$  are increased because of the additional input capacitance of  $G_c$ . If the wire capacitance of net  $A$  and  $B$  are  $CwA$  and  $CwB$ , the pin capacitance of input  $A$  and  $B$  of gate  $G$  are  $CinA$  and  $CinB$ , the delay increase can be calculated as:

$$\begin{aligned} \delta t_{G1} &= t_{G1}|_{\text{load}=CwA+2*CinA} - t_{G1}|_{\text{load}=CwA+CinA} \\ \delta t_{G2} &= t_{G2}|_{\text{load}=CwB+2*CinB} - t_{G2}|_{\text{load}=CwB+CinB} \end{aligned} \quad (4)$$



**Figure 3 Gate Cloning**

These timing changes should be compared against the available timing slack of the related timing paths. For example, if the minimum positive timing slack through  $N$  is  $\Delta t$ , the following condition has to be satisfied:

$$\Delta t \geq \max\{\delta r_{G1}, \delta r_{G2}\} - \delta r_G \quad (5)$$

Next, we use another example circuit to better illustrate how to estimate the softness changes. As shown in Figure 3(c), node  $N$ , having softness  $S_N(L_N, T_N, E_N)$ , drives a logic cone through three fanouts G1, G2 and G3. Suppose G3 is removed from the logic cone as a result of splitting this logic cone, we want to estimate the changes in softness of the affected nodes.

First, the relation between the logic factor of node  $N$  and its fanout nodes ( $D, H, F$ ) can be expressed as:

$$L_N = P_r^B(1) * L_D + L_H + P_r^C(0) * L_F \quad (6)$$

where  $P_r^B(1)$  is the probability for node  $B$  carrying a logic 1, the non-controlling value of a NAND gate, and  $P_r^B(1) * L_D$  is the probability for noise at  $N$  to propagate through gate G1;  $P_r^C(0)$  is the probability for node  $C$  carrying a logic 0, and  $P_r^C(0) * L_H$  is the probability for noise at  $N$  to propagate through gate G3. The single-input buffer G2 (BUF4) does not change the probability for propagation. Similarly, the logic factor after G3 is removed from the logic cone is given by:

$$L_N' = P_r^B(1) * L_D + L_H \quad (7)$$

Second, the timing factor change can be illustrated by Figure 4.  $T_N$  can be derived from the timing factors of its fanout nodes and the respective gate delays. Let  $T_D, T_H$  and  $T_F$  be the sensitive windows whose start and end times are:  $\{t_D^{start}, t_D^{end}\}, \{t_H^{start}, t_H^{end}\}$  and  $\{t_F^{start}, t_F^{end}\}$ , respectively. If the maximum and minimum delays of G1, G2 and G3 are  $\{d_{G1}^{min}, d_{G1}^{max}\}, \{d_{G2}^{min}, d_{G2}^{max}\}$  and  $\{d_{G3}^{min}, d_{G3}^{max}\}$ , respectively, the original noise sensitive window at node  $N$  has a start time and an end time:

$$(t_N^{start}) = \min\{t_D^{start} - d_{G1}^{max}, t_H^{start} - d_{G2}^{max}, t_F^{start} - d_{G3}^{max}\}$$

$$(t_N^{end}) = \max\{t_D^{end} - d_{G1}^{min}, t_H^{end} - d_{G2}^{min}, t_F^{end} - d_{G3}^{min}\}$$

And the noise sensitive window after removing G3 has a start time and an end time:

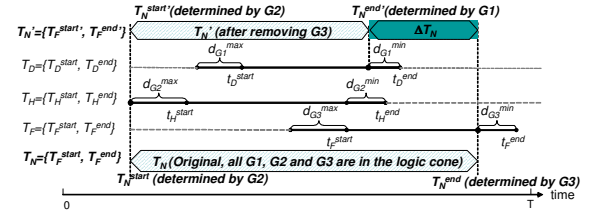
$$(t_N'^{start}) = \min\{t_D^{start} - d_{G1}^{max}, t_H^{start} - d_{G2}^{max}\}$$

$$(t_N'^{end}) = \max\{t_D^{end} - d_{G1}^{min}, t_H^{end} - d_{G2}^{min}\}$$

The old and new timing factors can be calculated as:

$$T_N = (t_N^{end}) - (t_N^{start}) \text{ and } T_N' = (t_N'^{end}) - (t_N'^{start})$$

Third, the change in the electrical factor  $E_N$  due to logic cone split is caused by the change of load capacitance. Originally,  $E_N = R_e^N(INVX1, C_{total})$ , with  $C_{total} = C_w + C_{in1} + C_{in2} + C_{in3}$ , where  $C_w$  is the wire capacitance and  $C_{in1}, C_{in2}, C_{in3}$  are input capacitances of the three fanout gates. After removing G3,  $E_N'$  changes to  $R_e^N(INVX1, C_{total}')$ , where  $C_{total}' = C_w + C_{in1} + C_{in2}$ .



**Figure 4 Timing window change**

In summary, the new softness after removing G3 from the original logic cone can be calculated as:

$$S_N'(L_N', T_N', E_N') = T_N' * L_N' * E_N' \quad (8)$$

A circuit node driving large logic cones may have many fanouts. If the number of fanouts is  $K$ , there are  $2^K$  ways to split the logic cone. A proper logic cone partition is crucial to the effectiveness of the gate cloning. We developed a procedure to search for a proper partition that optimally reduces  $S_N$  without causing timing violation, as described by the pseudo-code *SplitLogicCone* shown in Figure 5. The process *SplitLogicCone* has three arguments: the node  $N$ , a list of its fanout (*FanoutList*), and a target softness ( $S_{th}$ ). Since the logic cone can not be split at node  $N$  if it has only one load, the first step (Line 1-5) is to forward trace the circuit from  $N$  to the first node  $M$  with multiple loads. If a PO is reached during the search, gate cloning is not applicable.

```

SplitLogicCone (N, FanoutList, Sth)
1. /* Tracing forward from node N to the node with multiple fanout. */
2. while (sizeof(FanoutList) == 1)
3.   M = The only fanout;
4.   if (M is a PO) return (-1); /* gate cloning is not possible. */
5.   FanoutList = get_fanout(M);

6. NodeList1 = FanoutList; NodeList2 = {};
7. CurrentSn = Sn; TimingOk = 0; K = sizeof(FanoutList)

8. for i = 1 to K begin /* Outer Loop for logic cone split */
9.   MaxDeltaSn = 0;

10.  foreach ThisFanout in {NodeList1} begin
11.    TempNodeList1 = {NodeList1} - ThisFanout;
12.    TempNodeList2 = {NodeList2} + ThisFanout;
13.    if (check_timing(TempNodeList1, TempNodeList2) == 0)
14.      continue; /* Skip the rest if timing is not met */
15.    TimingOk = 1; /* At least one partition does not fail timing */
16.    Sn1 = estimate_softness(NodeList1);
17.    Sn2 = estimate_softness(NodeList2);
18.    if (Sn2 > Sth) continue; /* Should not create new soft spots */
19.    if (Sn1 <= Sth && Sn2 <= Sth)
20.      NodeList1 = TempNodeList1;
21.      NodeList2 = TempNodeList2;
22.      return; /* Softness reduction goal achieved, done */
23.    if (CurrentSn - Sn1 - Sn2 > MaxDeltaSn)
24.      MaxDeltaSn = CurrentSn - Sn1 - Sn2;
25.      CandidateSn = Sn1 + Sn2; CandidateNode = ThisFanout;
26.    End /* inner loop */

27. /* Update the partitions to result in the largest softness reduction */
28. FanoutList1 = {FanoutList1} - CandidateNode;
29. FanoutList2 = {FanoutList2} + CandidateNode;
30. CurrentSn = CandidateSn;
31. End /* outer loop */

32. if (TimingOk == 0) return (-1); /* No partition found, gate cloning failed */
33. /* Further logic cone split needed */
34. if (Sn1 > Sth) SplitLogicCone(M, NodeList1, Sn1, Sth);
35. if (Sn2 > Sth) SplitLogicCone(M, NodeList2, Sn2, Sth);

```

**Figure 5 Pseudo-code: SplitLogicCone**

Next, the process attempts to split the logic cone into two partitions (Line 6): *NodeList1*, which will be driven by  $N$ , is initialized to the original fanout list; and *NodeList2*, which will be driven by the cloned node  $N_c$ , is initialized

to an empty list. Then each iteration of the main loop (Line 8-31) attempts to move one node from *NodeList1* to *NodeList2*. The inner loop (Line 10-28) selects the node to be moved by evaluating the consequence of moving each remaining node in *NodeList1* to *NodeList2*: a move is invalid if it causes timing violations or the softness of the cloned gate to become larger than  $S_{th}$ ; if the resulting softness of both  $N$  and  $N_c$  are less than  $S_{th}$ , the process terminates successfully. If none of the above is true, a candidate move causing the largest softness reduction will be kept. As the inner loop terminates, both node lists are updated by moving the surviving candidate from *NodeList1* to *NodeList2* (Line 27-30). After the main loop is executed  $K$  times (the number of fanouts of node  $N$ ), the process returns a failure if none of the tried partition will meet timing; otherwise, if no partition can be found to meet the condition in Line 19, the process calls itself to further split the logic cone (Line 34-35).

Gate cloning can reduce the logic and timing factor. However, it does not help to reduce the electrical factor. Therefore it can not be applied to the soft spots whose high softness values are caused primarily by a large  $E_N$ .

#### 4. Cell Resizing

Cell resizing is to replace a logic gate by a functionally equivalent gate with difference size and driving strength. It can reduce the electrical factor and therefore the softness of a node. Based on the discussion in section 2, there are two ways to reduce  $E_N$ : driver downsizing and load upsizing. For example, reducing  $E_N$  in Figure 3(c) can be done by downsizing gate G from INVX4 to INVX2, or upsizing G3 from NOR2X2 to NOR2X4 or a combination of both. Both approaches have either positive or negative effects on the delays and softness of the affected circuit nodes. However, they do not affect the logic factors, and the effect on the timing factor is negligible. We now use this example to explain the effects of driver downsizing and load upsizing.

##### (1) Downsizing gate G: INVX4 $\rightarrow$ INVX2

The overall delay change has two part,  $\delta_G$  and  $\delta_{G0}$ . The  $\delta_G$  is the increased delay of gate G because the smaller INVX2 has longer delay than INVX4 under the same loading condition; and  $\delta_{G0}$  is the decreased delay of gate G1 due to a smaller input capacitance of INVX2 as compared to INVX4.

The electrical factor  $E_N$  is reduced from  $R_e^N(INVX4, C_{total})$  to  $R_e^N(INVX2, C_{total})$ , where  $C_{total} = C_w + C_{in1} + C_{in2} + C_{in3}$  is the total load driven by G. Note that  $E_A$ , the electrical factor of node A is increased as a side effect because of the reduction in  $C_A$ , the total load driven by gate G1. This change can be estimated similarly.

##### (2) Upsizing gate G3: NOR2X2 $\rightarrow$ NOR2X4

The overall delay change has two parts:  $\delta_G$  and  $\delta_{G3}$ . The  $\delta_g$  is the increased delay of gate G because its total load capacitance is increased due to a larger input capacitance of NOR2X4 as compared to NOR2X2; and  $\delta_{G3}$  is the decreased delay of gate G3 because NOR2X4 has shorter delay than NOR2X2 under the same load.

The electrical factor  $E_N$  is reduced from  $R_e^N(INVX4, C_{total})$  to  $R_e^N(INVX2, C_{total})$ , where  $C_{total}'$  is the total load of G after the upsizing,  $C_{total}' > C_{total}$  because of the large input capacitance of NOR2X4 as compared to NOR2X2. Note that the  $E_F$  is increased as a side effect from  $R_e^F(NOR2X2, Cl_F)$  to  $R_e^F(NOR2X4, Cl_F)$ , where  $Cl_F$  is the total load driven by gate G3.

From our experiences, driver downsizing is more effective than load upsizing. The softness reduction is more significant and the perturbation on other node is smaller. It can also reduce the total cell area.

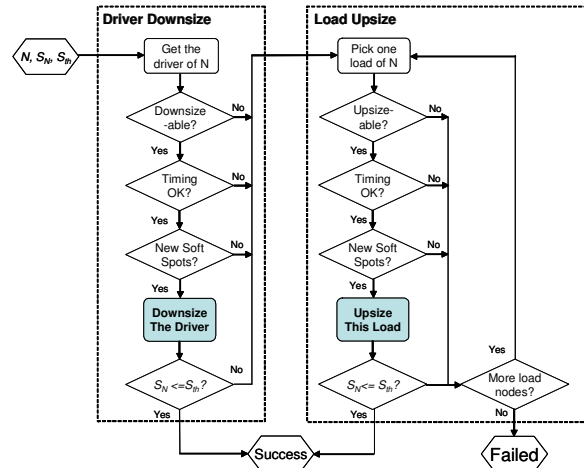


Figure 6 Cell Resizing flow

Figure 6 shows the process to search for the solution that requires minimal circuit change to achieve the desired softness reduction. Given a soft spot  $N$ , its softness  $S_N$  and a target softness value  $S_{th}$  ( $S_{th} < S_N$ ), the driver downsizing is first tried. Three conditions have to be met for the downsizing to be feasible: (1) the driver of  $N$  has to be “downsize-able”, i.e., a smaller gate has to exist in the cell library; (2) the timing requirement of related paths has to be satisfied; and (3) the softness values of all related nodes has to remain below  $S_{th}$ . The driver gate is then downsized if all three conditions are met. If the resulting  $S_N$  is reduced to below  $S_{th}$ , the process terminates successfully (“Done”). The load upsizing process is entered when any of the three conditions is not satisfied or the driver downsizing is feasible but  $S_N$  is still larger than  $S_{th}$ . It is similar to the driver downsizing and go through every load of  $N$ . The process terminates successfully as soon as the softness reduction goal is reached. If  $S_N$  can not be reduced to below  $S_{th}$  when all load gates have been processed, the process terminates with a “Failed” flag.

#### 5. Robustness Closure

A technique aiming at SET tolerance enhancement must have no negative impact on the design’s timing closure and physical layout. Otherwise, the circuit changes will incur further timing or layout problems that can only be resolved by more design iterations. For example, [4] did not consider realistic design constraints, such as the driving strength and wire capacitances. As a result, when applied to a chain of identical inverters, the algorithm

returned a resizing scheme of a chain of inverters with monotonously decreasing sizes (40x-18x-8x-4x-2x-1x), which most likely will not be the best structure in a real design. In addition, the best way to maximize their efficiency is to integrate them with the design flow.

As shown in Figure 7, a typical digital design process starts with the “Front-end Design”, including logic/circuit design and functional verification, followed by the “Back-end Design”, including logic synthesis, physical layout and timing closure. Its primary objective is to produce a functioning design that meets certain speed goal with minimum area. As the noise-tolerance is not an optimization criterion, certain circuit elements in the timing-closed design might be still highly vulnerable. Therefore, an additional optimization step (“Robustness Closure”) to fix these vulnerabilities is necessary. We can reuse the existing analysis results obtained from the other steps in the flow (such as static timing analysis results, extracted RC information, etc.). The RObustness Compiler (ROCO) was constructed based on these ideas.

ROCO is implemented in TCL environment with its key engines written in C. It can be directly executed from TCL-based STA tools, such as PrimeTime. This seamless interface can improve its performance as it not only takes advantage of the speed and accuracy of commercial tools, but also provides direct feedback to these tools.

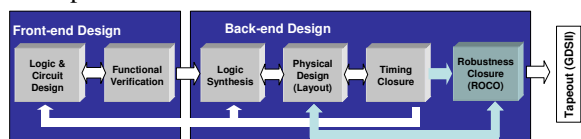


Figure 7 Robustness Closure in design flow

As shown in Figure 8, ROCO requires three inputs: a “Timing-Closed Design”; a “Design Analysis Report”, which contains information of the design such as timing, area and RC parasitic; and a “Robustness Specification” which defines the desired robustness level using a “soft spot ratio” ( $R_S$ ). The heuristic  $R_S$  specifies a certain percentage of circuit nodes with highest softness values as soft spots and determines the threshold  $S_{th}$ . The goal is to reduce softness of all circuit nodes to below  $S_{th}$ . The “Robustness Specification” also provides the allowed timing ( $\Delta T$ ) and area ( $\Delta A$ ) overhead. The  $\Delta T$  will allow ROCO to increase the system clock period from  $T$  to  $T + \Delta T$  in order to meet the robustness specification.  $\Delta T$  can be zero if speed degradation is not acceptable. The  $\Delta A$  specifies the maximum allowed area increase. Determining  $R_S$ ,  $\Delta T$  and  $\Delta A$  is beyond the scope of this work.

Given all these inputs, ROCO first uses a “Robustness Estimation Engine” based on the soft spot analysis to evaluate the softness of all circuit nodes. Then it generates a collection of soft spots with  $S_N > S_{th}$ . Next, the “Softness Reduction Engine” uses the “Gate Cloning” and “Cell Resizing” engines to reduce the softness of the identified soft spots. Gate cloning is applied first because it ensures convergence of the methodology: the softness reduction achieved by gate cloning will not be reversed by the cell resizing. These processes are concurrently checked by the “Constraint Monitor Engine” to ensure that no design

constraint is violated. Finally ROCO generates a “Robustness-Closed Design”.

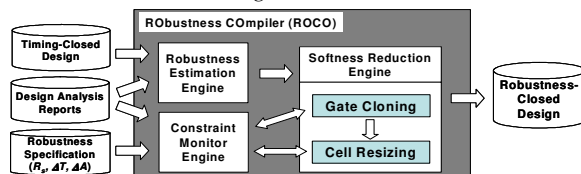


Figure 8 RObustness Compiler (ROCO)

## 6. Experimental Results

In this section, we will demonstrate the efficiency and discuss some distinguished features of ROCO via several experiments.

First we will verify that both techniques can effectively reduce improve circuit SET tolerance when individually applied on the design by HSPICE simulations. Due to the speed and capacity limitations of HSPICE, we only used small circuits to prove the concept. Each experiment circuit was timing-closed, post-layout netlist. We ran ROCO and obtained the modified design after gate cloning (CKT-gc) and cell resizing (CKT-cr). We then ran HSPICE simulations on CKT, CKT-gc and CKT-cr. During the simulation, a large number of glitches with random shape and timing were injected only to the affected nodes. The number of errors observed at circuit POs were counted and compared among all three circuits.

We used two experiment circuits: CKT1 with 73 nodes and CKT2 with 192 nodes.  $R_S$  was set low (CKT1: 8%, CKT2: 2%) to limit the number of soft spots to reduce the simulation time. Among the 6 soft spots in CKT1, 5 nodes were cloned and 4 cells were resized; among the 4 soft spots in CKT2, 4 were cloned and 2 cells were resized. Table 1 and Table 2 shows the results: the column “Glitch” lists the number of glitches injected during the HSPICE simulation; the column “Error” lists the number of errors observed at the POs; and the column “E.R.” is the error rate (Error/Glitch). As we can see in the last column, both the gate cloning and cell resizing were able to reduce the error rate significantly (from 41% to 61%).

Next we will prove that given realistic constraints, ROCO can effectively and economically improve circuit reliability. In addition to CKT1 and CKT2, we also used one large circuit CKT3 with 3156 circuit nodes.

Table 1. HSPICE simulation: Gate Cloning

	Original Circuit (CKT)			Post Gate Cloning (CKT-gc)			Error Reduction
	Glitch	Error	E.R.	Glitch	Error	E.R.	
CKT1	25600	2464	9.6%	66560	3124	4.7%	51%
CKT2	2048	388	18.9%	4096	297	7.3%	61%

Table 2. HSPICE simulation: Cell Resizing

	Original Circuit (CKT)			Post Cell Resizing (CKT-cr)			Error Reduction
	Glitch	Error	E.R.	Glitch	Error	E.R.	
CKT1	32768	4439	13.5%	32768	2004	6.1%	55%
CKT2	8192	823	10.0%	8192	483	5.9%	41%

Table 3 lists the robustness specifications: the number of nodes (2<sup>nd</sup> col.), the soft spot ratio  $R_S$  (3<sup>rd</sup> col.) and the

threshold  $S_{th}$  (4<sup>th</sup> col.), the clock period  $T$  (5<sup>th</sup> col.) and the allowed timing overhead  $\Delta T$  (6<sup>th</sup> col.). We artificially set  $R_s$  of the two small circuits to 20% and  $R_s$  of the large CKT3 to 2%. Because timing is usually a more important performance concern than area, we set the  $\Delta T$  to be 0 and relaxed the area overhead constraints.

Table 4 shows the experiment results: for each circuit the three rows lists the data of the original circuit (Orig.), circuit after gate cloning (G.C.) and cell resizing (C.R), respectively. The five columns show the total softness  $S_{total}$ , the maximum softness  $(S_N)_{max}$ , the number of identified soft spots  $N_{ss}$  (the number of nodes whose softness values is larger than  $S_{th}$ ), the total softness of the soft spots  $(S_N)_{ss}$ , and the area with percentage change. We now explain these experiment results.

**Table 3 Robustness specifications**

	Nodes	$R_s$	$S_{th}$	$T(ns)$	$\Delta T(ns)$
CKT1	73	20%	1.75	1.8	0
CKT2	192	20%	1.09	2.4	0
CKT3	3156	2%	3.2	8.0	0

**Table 4 ROCO execution results**

		$S_{total}$	$(S_N)_{max}$	$N_{ss}$	$(S_N)_{ss}$	Area ( $\mu m^2$ )
CKT 1	Orig.	53.88	6.24	15	27.36	2039
	G.C.	50.26	3.24	4	9.30	2175(+6.7%)
	C.R.	45.59	2.08	2	3.95	2095(+2.7%)
CKT 2	Orig.	73.17	2.75	38	34.31	5172
	G.C.	72.14	1.38	8	8.5	5415(+4.7%)
	C.R.	68.87	1.04	0	0	5252(+1.5%)
CKT 3	Orig.	1623	46.62	63	499	83915
	G.C.	1441	10.3	12	72.1	85645(+2.1%)
	C.R.	1408	5.8	6	22.9	85269(+1.6%)

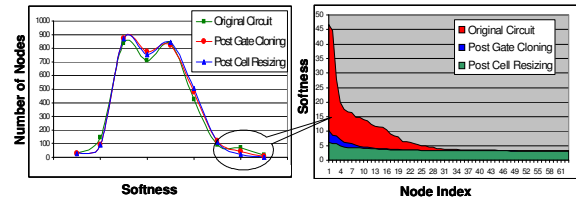
First, we proved that a small number of nodes with high softness values contribute to the majority of the circuit vulnerability. For example, the top 2% nodes (63) in CKT3 make up to 31% (499/1623) of the total softness.

Second, ROCO can significantly reduce the number of soft spots and their softness: the number of soft spots ( $N_{ss}$ ) in CKT1 was reduced from 15 to 4 after GC and to 2 after CR; in CKT2,  $N_{ss}$  was reduced to 8 after GC and all soft spots were eliminated after CR. For CKT3,  $N_{ss}$  was reduced to 12 and 6, respectively; and  $(S_N)_{ss}$  was reduced by 95.4% (499 to 22.9).

Third, the total softness  $S_{total}$  only slightly decreased (e.g. by 13%, from 1623 to 1408, in CKT3), because the softness of some neighboring nodes may be increased. We can view it as “softness redistribution”, as illustrated in Figure 9. On the left, the x-axis is the softness values in log-scale and the y-axis is the number of nodes with different softness. The top-ranked soft spots correspond to the small tail toward the right end, which shrinks after each step. The number of nodes with intermediate softness values increases slightly. The right graph zooms in on the 63 soft spots on the right tail, where the actual softness (y-axis) is plotted for all circuit nodes (x-axis). It clearly shows how much the softness of the soft spots is reduced.

Fourth, it is not always possible to eliminate all soft spots under given constraints and budgets. In the end, 2 nodes in CKT1 and 6 nodes in CKT3 still have softness values above  $S_{th}$ . For further improvement, more design

overhead should be allowed. Finally, given all required inputs, ROCO finished within ~23 seconds for the small CKT1 and CKT2; for the large CKT3, the runtime is ~678 seconds. ROCO is fast because (1) it is static and does not need dynamic simulation; (2) it takes advantage of the high performance of commercial static timing analysis tool; and (3) it only focus on a limited number nodes.



**Figure 9 Softness redistribution in CKT3**

## 7. Conclusion

In this paper, we presented a robustness closure engine “RObustness COmpiler” that can efficiently improve circuit reliability using two localized circuit modification techniques: gate cloning and cell resizing. Seamlessly integrated with existing design flow, it will greatly facilitate the design of reliable nanometer circuits.

## 8. References

- [1] S.E. Kerns, et al. “The design of radiation-hardened ICs for space: a compendium of approaches,” Proc. of the IEEE, Vol. 76, No. 11, pp. 1470-1509, Nov. 1988.
- [2] Anghel, L., Nicolaidis, M., “Cost Reduction and Evaluation of a Temporary Faults Detecting Technique,” Proc. of Design, Automation and Test in Europe Conference, pp. 591-598, March 2000.
- [3] Y.Zhao, S.Dey, “Separate Dual Transistor Register-an Circuit Solution for on-line Testing of Transient Errors in UDSM-IC,” in Proc. Intl. On-line Testing Symposium 2003, pp.7-11, June 2003.
- [4] Dhillon, Y.S., Diril, A.U., Chatterjee, A., Singh, A.D., “Sizing CMOS circuits for increased transient error tolerance,” in Proc. of Intl. On-line Testing Symposium, pp. 6-10, Madeira Island, Portugal, June 2004.
- [5] Quming Zhou, Kartic Mohanram, “Transistor Sizing for Radiation Hardening”, Intl. Reliability Physics Symposium, pp. 310-315, 2004.
- [6] Yuvraj S. Dhillon, Abdulkadir U. Diril, Abhijit Chatterjee, Cecilia Metra, “Load and Logic Co-Optimization for Design of Soft-Error Resistant Nanometer CMOS Circuits,” in Proc. of Intl. On-line Testing Symp., pp. 35-40, 2005.
- [7] C.Zhao, X. Bai, S.Dey, “A scalable soft spot analysis methodology for compound noise effects in nano-meter circuits,” DAC’04, pp. 894-899, June 2004.
- [8] O. Coudert, R. Hadad, “New algorithms for gate sizing: A comparative study,” DAC’96, pp. 734-739, June 1996.
- [9] A. Srivastava, C. Chen, and M. Sarrafzadeh, “Timing driven gate duplication in technology independent phase,” ASP-DAC’01, pp. 577-582, Jan. 2001.
- [10] C. Zhao, Y. Zhao, S. Dey, “Constraint-Aware Robustness Insertion for Optimal Noise-Tolerance in VLSI Circuits,” DAC’05, pp. 190-195, June 2005.
- [11] S. Mitra, N. Seifert, M. Zhang, Q. Shi and K.S. Kim, “Robust System Design with Built-In Soft Error Resilience,” IEEE Computer, Vol. 38, No. 2, pp. 43-52, Feb. 2005.