Charge Sharing Alleviation and Detection for CMOS Domino Circuits

Shih-Chieh Chang, Ching-Hwa Cheng, Wen-Ben Jone**, Shen-Da Li, and Jinn-Shyan Wang*

Department of Computer Science and Information Engineering
*Department of Electrical Engineering
National Chung Cheng University
Chiayi, Taiwan, Republic of China

**Department of Computer Science
New Mexico Tech
Socorro, NM 87801
U. S. A.

{scchang, chc, jone, sdl87@ cs.ccu.edu.tw, ieegsw@ccunix.ccu.edu.tw}

Abstract

Charge sharing which occurs in any CMOS domino gate may degrade the output voltage level or may even cause an erroneous output value. In this paper, this problem is thoroughly investigated by considering circuit topology and circuit function. We describe a method to measure the sensitivity (called CS-vulnerability) of the charge-sharing (CS) problem for each domino gate. A method to derive the CS-vulnerability and the test vector for each domino gate is suggested. We also propose a transistor reordering method to dramatically reduce the CS-vulnerabilities for all domino gates, so that the CS problem can be alleviated. We also prove theoretically that a set of test vectors generated for single CS faults can also detect all multiple CS faults. This good property significantly guarantees the test quality for the CS faults of domino circuits.

Index terms: *domino circuit, charge sharing, CS-vulnerability, transistor reordering, multiple faults.*

1. Introduction

Unlike a CMOS static logic gate where complementary functions are implemented in N and P transistors, an N-type domino gate requires only the N transistors to perform the pull-down function and uses a pre-charge clocked pMOS to perform the pull-up function. Due to the saving of P-transistors, domino logic offers significantly higher speed and smaller area than static CMOS circuits. As a result, domino logic is extensively used in today's high-performance processors [1], [3], [4], [15]. A typical domino AND gate design is shown in Figure 1.

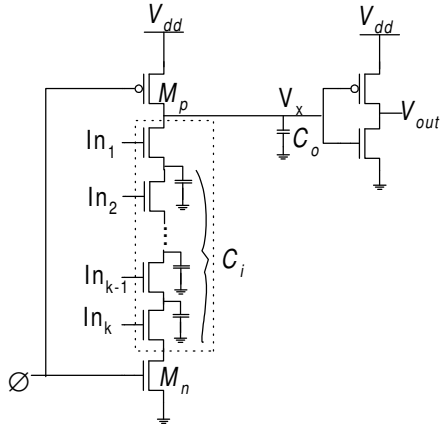

Figure 1. A CMOS domino AND gate.

Domino gates have two basic operating phases that are the *pre-charge* phase when the clock ϕ is low, and the evaluation phase when the clock ϕ is high. During the pre-charge phase, the capacitance C_o will be charged to high. During the *evaluation* phase, if all the inputs In_i to N-transistors are high, the voltage V_x is pulled to ground. Despite several advantages, domino circuits require special care to avoid the charge-sharing problem that may result in erroneous output values [7], [12]. The charge-sharing problem is termed as the ***CS*** problem in the following discussion. Consider the domino AND gate in Figure 1. During the evaluation phase, suppose all inputs In_i are high except input In_k located next to the clocked transistor M_n is low. Since In_k is low, ideally, the value of V_x should remain high; however, the charge in C_o loaded in the pre-charge phase is shared or re-distributed to the (source-drain) junction capacitance of those turned-on transistors. Let the equivalent value for the junction capacitance be C_i as shown in Figure 1. The output voltage V_x becomes V_{dd}* $C_o/(C_o+C_i)$ in the evaluation phase. If C_i is large enough, V_x may become too low to turn on the N-transistor of the subsequent inverter. As a result, under certain input conditions, charge re-distribution among junction capacitance at internal nodes can cause error or glitch at outputs.

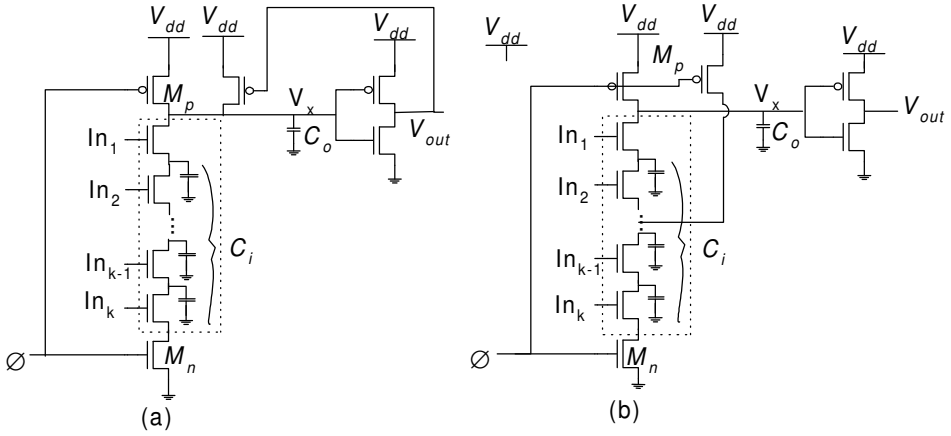

Figure 2. (a) Addition of a weak pMOS pull-up transistor. (b) Internal node pre-charging.

Several researchers have attempted to solve the CS problem in domino logic design. For example, a weak pMOS pull-up transistor can be added to the output of a domino gate shown in Figure 2(a) [6], [10], [14]. The weak pull-up device in the feedback loop can be used to provide additional current source, so as to prevent the loss of output voltage level due to the CS problem. The weak pull-up logic, however, is sensitive to the pull-up and pull-down ratio of transistors and therefore is sensitive to process variation. Another way of preventing the CS problem is to pre-charge internal nodes as shown in Figure 2(b) [6], [10], [14]. Though the CS problem can be alleviated, both solutions can incur area overhead as well as circuit performance degradation. Normally, the size of the weak pull-up P-transistor depends on the number of N-transistors on the domino gate. In today's domino circuit design, because the number of N-transistors can be large, one may require a large-size pull-up transistor and, as a result, performance degradation can be very severe. Further, faults occurring at the weak pull-up loop or the internal node pre-charge circuit may cause CS faults. Therefore, CS fault detection for domino circuits is necessary.

Let us further analyze the CS problem by considering the circuit in Figure 1 again. For the CS problem **not** to occur, the voltage $V_x = V_{dd}*C_o/(C_o+C_i)$ must be greater than the threshold voltage V_{thresh} of the inverter, i.e., $V_x = V_{dd}*C_o/(C_o+C_i) > V_{thresh}$. Among the variables, the value of C_o can be obtained by layout parameter extraction, while V_{thresh} can be determined from the aspect of reliable circuit design. After obtaining C_o and V_{thresh}, one can easily calculate the threshold value of C_i called $C_{ithresh}$. Thus, we have $C_{ithresh} = (V_{dd}*C_o/V_{thresh}) - C_o$, and the CS problem will not occur if C_i is smaller than $C_{ithresh}$. On the other hand, the major contributor of C_i is the junction capacitance of consecutive turn-on transistors next to the output inverter. Therefore, the value of C_i depends on the input conditions to turn on/off N-transistors (Note that there may exist no input pattern to turn on all In_i's except In_k). Among all

possible input patterns, assume that C_{imax} is the largest C_i so that the CS fault can be observed at primary outputs. We define the *CS-vulnerability* of a domino gate as $\boldsymbol{C_{imax}/C_{ithresh}}$. The idea of CS-vulnerability is to describe the degree of sensitivity for a domino gate to have the CS problem. When the CS-vulnerability of a domino gate is larger than 1, i.e., $C_{imax} > C_{ithresh}$, the CS problem for the domino gate can occur for some input patterns. If the CS-vulnerability is smaller than 1, but close to 1, the CS problem is also likely to happen, because process variation may change the values of C_i and C_o. The advantages for finding the CS-vulnerability for each gate are two fold. First, for circuit designers, the CS-vulnerability of a gate gives hints about whether the gate needs special care for preventing the CS problem. Secondly, for test engineers, it is necessary to find test vectors to test domino gates with large CS-vulnerability values.

In this paper, first, we give the basic concept to compute the value of C_{imax} for each domino gate so that its CS-vulnerability can be derived. Since the value of C_{imax} depends on the number of transistors that can be consecutively turned on, it relies on whether there exists an input pattern to simultaneously turn on/off certain transistors. As a result, the value of C_{imax} depends on the transistor ordering for the domino gate. Under some timing constraints, we then propose a transistor-reordering algorithm to reduce the value of C_{imax}. Note that the reduction of C_{imax} is very useful since the smaller the value of C_{imax} is, the smaller the CS-vulnerability will be. In the case of using a weak pull-up transistor to alleviate the CS problem for each domino gate, a smaller value in C_{imax} allows one to use a smaller size of (or even none) weak pull-up transistor that leads to both the area/speed gain. Finally, we model each CS fault as a single stuck-at-1 fault to generate the corresponding test vector for the CS fault (and also the corresponding CS-vulnerability) of each domino gate. We also find a good property of domino circuits in that multiple CS fault masking is non-existent. Thus, any set of test patterns for all single CS faults also detects all possible multiple CS faults.

Section 2 describes the concept of CS-transistors in detail, and discusses the computation of CS-vulnerability. The transistor reordering technique is then applied in Section 3 to reduce the value of CS-vulnerability for each domino gate. In Section 4, we develop the fault model, and the theorem of detecting multiple CS faults using single CS fault test vectors is established. The automatic test pattern generation (ATPG) for CS faults is also given in this section. Experimental results obtained by simulating a set of MCNC benchmark circuits are presented in Section 5. Finally, Section 6 concludes this paper.

2. Charge Sharing and CS-vulnerability

We model the degree of sensitivity for the CS problem of a domino gate by the CS-vulnerability whose value has been defined as $C_{imax}/C_{ithresh}$. To obtain the CS-vulnerability of a domino gate, the value of $C_{ithresh}$ can be estimated during the design of the domino gate, but the value of C_{imax} depends on the entire circuit structure and function.

First, we would like to mention that the CS problem for a domino gate can be affected by vectors applied in the previous cycles. That is, the CS problem also depends on the previous states of the domino gate. Consider the circuit in Figure 3 and assume that two vectors (*a b c, d e f*) = {(100, 100), (110, 110)} are applied to the domino gate consecutively. After applying the first vector (*a b c, d e f*) = (100, 100), we have charge Q_1 stored in the parasitic capacitance of transistors *Ta* and *Td*. Note that the voltage value is equal to the total charge divided by the total capacitance. During the evaluation phase of applying the second vector, the total charge Q includes the residual charge Q_1, and the pre-stored charge Q_2 in C_o, i.e., $Q = Q_1 + Q_2$. The total capacitance C, to which charge Q will be shared or redistributed, includes capacitance C_o and the parasitic capacitances of four turn-on transistors (*Ta Tb, Td Te*). Because of $V_x = (Q_1+Q_2)/C$, if there is more residual charge of Q_1, then the voltage in V_x is larger (i.e., V_x drops less) and the CS problem will be less severe.

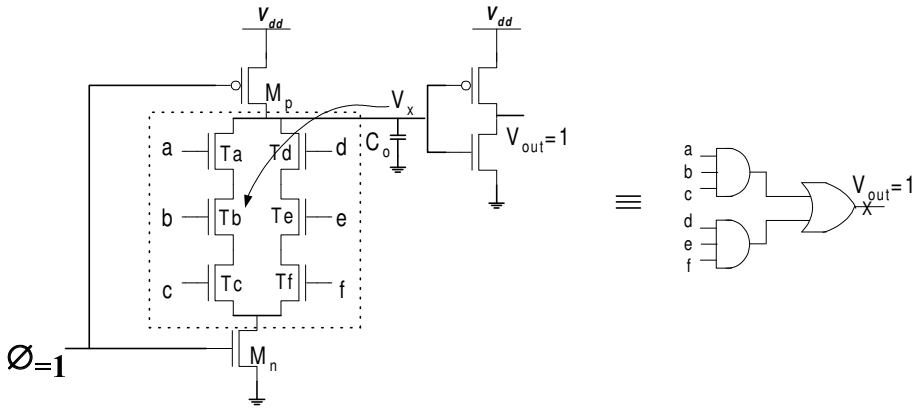

Figure 3. The direct and single-cycle charge cleaning.

The previous example shows that the CS problem depends on both the amount of residual charge left in the previous cycles and the amount of parasitic capacitance sharing the total charge in the current evaluation cycle. In order to derive the worst case of the CS problem, we analyze the CS problem by two stages: the *cleaning* stage and the *sharing* stage. The purpose is to find the minimum amount of residual charge

in the cleaning stage, and then to find the maximum amount of parasitic capacitance in the sharing stage. Although it requires two stages, in Subsection 2.1, we will show that practically the chance of "cleaning" the residual charge of a domino gate is very high. In fact, from the viewpoint of a safe design, it is reasonable to ignore the residual charge (i.e., the worst case), so that the CS-vulnerability only depends on the parasitic capacitance. Note that, when the domino circuit is tested, input patterns must be applied to release the residual charge to test the worst-case CS faults.

2.1 The cleaning stage

For a domino gate, as discussed above, the residual charge left by the previous clock cycles can reduce the occurring probability of the CS fault. In this subsection, we will present the idea of discharging operations by one cleaning vector or by a sequence of cleaning vectors. From these operations, one can know that there are enormous ways to clean the residual charge for the purpose of testing the worst-case CS faults. Based on the discharging path topology, the residual charge releasing process can be categorized as *direct cleaning* and *loop-back* cleaning.

- Direct cleaning: This is to directly release the residual charge by each column of transistors. Consider the example in Figure 3. If vector (*a b c , d e f*) = (x11, x11) is applied, the residual charge can be conducted to the ground through paths *Tb ->Tc ->Mn* and *Te->Tf ->Mn* where *x* stands for the flexibility of being either 1 or 0.

- Loop-back cleaning: This is to release the charge in a loop-back manner. Consider the example in Figure 4. If test vector *(a b c, d e f, g h i*) = (111, 110, 001) is applied, transistors (*Ta, Tb, Tc, Td, Te*) are turned on and the residual charge along path *Te->Td->Ta->Tb->Tc->*M_n in Figure 4 can be released. Similarly, test vector (*a b c, d e f, g h i*) = (110, 111, 001) can achieve another loop-back cleaning.

Further, based on the number of clock cycles required, the residual charge releasing process can be classified as *single-cycle* cleaning and *multi-cycle* cleaning.

- Single-cycle cleaning: In this case, the residual charge can be released in a single clock cycle regardless of using direct cleaning or loop-back cleaning. The example given in the direct cleaning method shown above is a single-cycle cleaning process. Test vector *(a b c, d e f, g h i)*=(111, 110, 110) is a single-cycle cleaning pattern using the loop-back cleaning method.

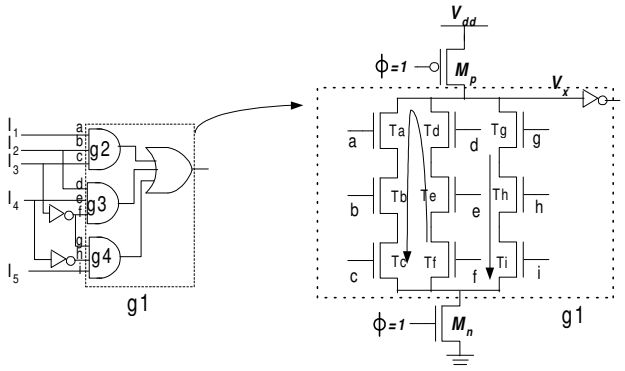

Figure 4. The loop-back and multi-cycle charge cleaning.

- Multi-cycle cleaning: To clean up the residual charge, it can also be done by several clocks cycles, i.e., the cleaning stage can be partitioned into several phases, and each of the phases will release some of the remaining charge. Once the charge of a transistor T_i is conducted to the ground, it must be guaranteed that T_i will not be charged again in the following phases. For example, in Figure 4, we can discharge the first-column transistors (*Ta Tb Tc*) by the cleaning vector (*a b c, d e f, g h i*) = (x11, xxx, xxx). The second cleaning vector (*a b c, d e f, g h i*) = (0xx, x11, xxx) can be used for discharging the second-column transistors (*Td Te Tf*). The third cleaning vector is (*a b c, d e f, g h i*) = (0xx, 0xx, x11) for discharging the third-column transistors (*Tg Th Ti*). So, domino gate *g1* requires three vectors to fulfill the cleaning process in this case. In fact, direct cleaning, loop-back cleaning, or a combination of both can achieve multi-cycle cleaning. In Figure 4, the residual charge can also be released by a combination of both direct cleaning and loop-back cleaning using two clock cycles. The first vector (*a b c, d e f, g h i*) = (111, 110, xxx) can be used for discharging the first-column and second-column transistors *(Ta, Tb, Tc, Td, Te, Tf)* by discharging path *Te->Td->Ta->Tb->Tc->*M_n. The second vector (*a b c, d e f, g h i*) = (0xx, 0xx, x11) can be used for discharging the third-column transistor *(Tg, Th, Ti)* by discharging path *Th->Ti->*M_n. Notably, the second cleaning vector prevents transistors *(Ta, Tb, Tc, Td, Te, Tf)* from being pre-charged again.

2.2 The sharing stage

From the previous subsection, we know that it is not too difficult to clean up all residual charge by some input patterns. For the CS problem, we only need to consider the amount of parasitic capacitance in the sharing stage by neglecting the effect of residual charge (i.e., the worst case). The reason comes from the fact that, during testing, the residual charge can be released by certain test patterns using the cleaning techniques discussed in Section 2.1. We now discuss how to find the maximum amount of parasitic capacitance for each domino gate. Among all possible input vectors, the largest value of C_i is called C_{imax}.

Consider the example in Figure 5. The worst-case CS problem at *g1* occurs if charge stored in the pre-charge phase is shared by the parasitic capacitance of transistors (*Ta, Tb, Tc, Te, Tf, Tg*). To realize this worst case, there must exist an input pattern ***v*** to activate the CS fault. Additionally, input pattern ***v*** must cause the CS fault effect to be observable at primary outputs. Due to the need to simultaneously satisfy the activation condition *((a b c d, e f g h)*=(1110, 1110*))* and the observation condition (I_5=1), there might exist no input pattern to realize the worst-case condition. In comparison to the cleaning stage, finding input vectors for the cleaning stage is much easier, because the cleaning stage does not need to consider the observation condition and also can be done in multiple clock cycles.

Without loss of generality, we assume that each transistor has the same parasitic capacitance. However, in real applications, the parasitic capacitance of each transistor in a domino circuit can be extracted from the circuit layout. Let us define those transistors that are consecutively turned on and next to the inverter stage be the *CS-transistors*, under the condition that there exists no conducting path at each transistor column. For example, in Figure 5, if there exists an input pattern to generate (*a b c d, e f g h*) =(1110, 1110), we say that transistors (*Ta, Tb, Tc, Te, Tf, Tg*) are CS-transistors. These CS-transistors will contribute their parasitic capacitance to C_i. So, computing the value of C_{imax} is analogous to computing the largest number of CS-transistors.

For a domino gate, one can find that the largest number of CS-transistors is equal to the total number of transistors minus the "width" of the domino gate. The width of a domino gate is defined as the number of parallel transistor columns. Such a situation is called the *structural worst case*. For example, in Figure 5, the structural worst case for the CS problem of domino gate ***g1*** is that both transistors *Td* and *Th* are turned off and the charge is shared among the parasitic capacitance of transistors {*Ta, Tb, Tc, Te, Tf, Tg*}. In this case, the number of CS-transistors is 6.

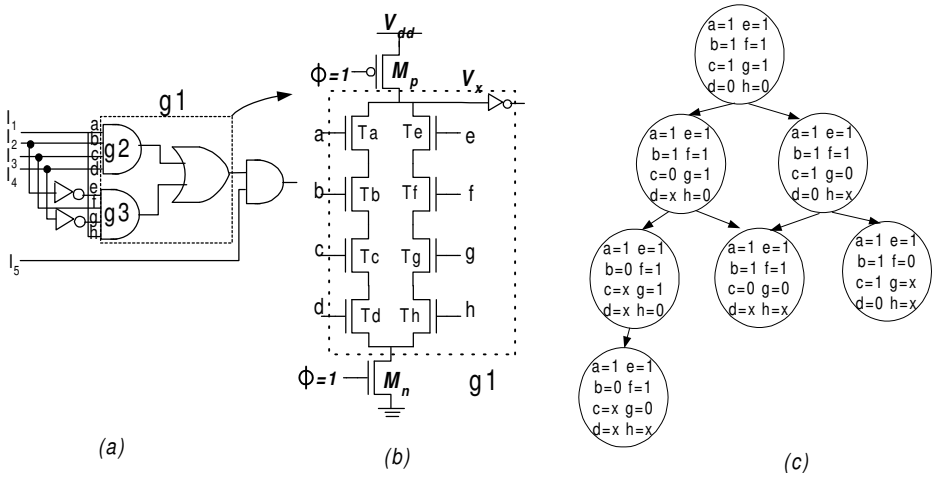

Figure 5. A search example of the CS state tree.

The structural worst case may not happen for some gates. Consider the example in Figure 5. The structural worst case is that transistors {*Ta, Tb, Tc, Te, Tf, Tg*} are turned on, while transistors {*Td, Th*} are turned off. To accomplish this goal, we must assign (*a b c d, e f g h*) = (1110, 1110), and this assignment can be shown to be impossible because of conflict in assigning *b*=1 and *e*=1.

Now, our objective is to find the largest number of CS-transistors that can be turned on for each domino gate. We adopt a tree-searching method that attempts to find a test vector to turn on the largest number of CS-transistors. Our algorithm starts from the structural worst case of a domino gate. If the structural worst case cannot be generated, it then selects another CS state with one less CS-transistor than the structural worst case in the CS state tree (Figure 5(c)). The process is repeated until we find a CS state that has a test vector.

We now use an example to illustrate our tree-searching algorithm. Consider again the example of Figure 5 whose searching tree is shown in Figure 5(c). The exhaustive search starts from the structural worst case by assigning (*a b c d, e f g h*) = (1110, 1110) for which there exist no test vector. Since there are six CS-transistors in the structural worst case, we continue to check whether it is possible to have a set of five CS-transistors. There are two different cases shown in the second level of the tree in Figure 5(c). We first check whether there exists a vector for the assignment (*a b c d, e f g h*) = (1110, 110x), and then the assignment (*a b c d, e f g h*) = (110x, 1110). Obviously, both assignments are impossible because of the test conflict between inputs (*a b c d*) and inputs (*e f g h*). Then, we check whether it is possible to have four CS-transistors, and the vectors for assignments (*a b c d, e f g h*) = (1110, 10xx), (10xx, 1110) and (110x, 110x) are also non-existent. Thus, we try to find a test vector for three CS-transistors. Since there indeed exists a test vector for the assignment (*a b c d, e f g h*) = (10xx, 110x), our algorithm terminates and concludes that the largest number of CS-transistors is 3. By applying our tree-searching method, we can obtain the largest number of CS transistors for each domino gate. As will be discussed in Section 4, the largest number of CS-transistors can be derived using an ATPG process with the aid of the search tree. In fact, the ATPG process will give both the largest number of CS-transistors and the corresponding test vector for each domino gate simultaneously.

Though a domino circuit can have a large number of transistors, the tree-searching method may not be as bad as it seems. The reasons are as follows. First, our algorithm terminates once a test vector, which results in the largest possible CS-transistors, is found. In other words, our algorithm does not have to “really” go through all possible

CS states, even though the algorithm may be exhaustive in the worst case. Secondly, even in the worst case, the number of iterations is not too bad. Consider a large domino gate with 30 transistors that are structured by 3 parallel columns (OR) of 10 serially chained transistors (AND) (at each column). The worst number of iterations is 999 using the above tree-searching algorithm. If a domino gate has 40 transistors with 4 columns of 10 transistors (at each column), the worst number of iterations is 9999. Table 1 summarizes these cases.

Table 1. The worst case search times of CS states.

# of parallel columns	# of serial transistors	Total CS states #
2	20	180
3	30	999
4	40	9999

3. Minimization of CS-vulnerability by Transistor Reordering

Consider again the domino gate in Figure 5. If the number of CS-transistors is 5, for example, there must exist an input vector to generate either one of the two assignments (*a b c d, e f g h*) = {(1110, 110x), (110x, 1110)}. The existence of such an assignment depends on both the circuit function and the transistor ordering. In this section, we discuss how to reorder the transistors of a domino gate so that the number of CS-transistors is minimized. We would like to reiterate that minimization of the CS-transistors implies minimization of C_{imax} (and thus the CS-vulnerability), if the size of each transistor is the same. We first propose a structured algorithm that guarantees to obtain an optimal solution. For some large circuits, the structured algorithm can be time-consuming. To reduce the CPU runtime, we then propose a heuristic method based on the signal probability of each node.

3.1 A Branch-and-Bound Algorithm for Optimal Solution

Let a domino gate with *n* parallel columns (OR) of *m* serially chained transistors (AND) at each column be denoted as an (*n, m*) domino gate. For example, the circuit in Figure 5 is a (2, 4) domino gate. A naive method will exhaustively examine $(m!)^n$ different kinds of transistor orderings, and the number of CS-transistors for each order must be computed using the search algorithm discussed in the previous section. Instead of blindly going though all possible orders, our structured algorithm adopts a branch-and-bound approach to prune many unnecessary ordering structures.

Let the number of CS-transistors for the structural worst case be $k\ (=(m-1)\times n)$. In the i^{th} iteration, our algorithm checks whether there exists a transistor order that can generate less than *k-i* CS-transistors. The key success of this algorithm is that, instead of going through the entire ordering domain, we only check some representative

orders without losing the optimality. If a representative order can generate *k-i* CS-transistors, then we do not consider this representative order (and all other orders associated with it) in the next iteration. This greatly reduces the required CPU time.

We illustrate this idea using the circuit in Figure 6 whose original order is (*a b c d, e f g h*). First, in the 0^{th} iteration of our algorithm, we want to know whether there exists an order which allows smaller than 6 (=*k*) CS-transistors. Note that the 0^{th} iteration is to consider the structural worst case. In this iteration, (*a b c d, e f h g*) can be used as a representative for *Order_SET* ={*(abcd, efhg), (abcd, ehfg), (acbd, efhg), (acbd, fheg), (abcd, hfeg), (acbd, ehfg), (bcad, fheg), (bcad, ehfg), (bcad, efhg)..}*. It can be found that each member in *Order_SET* has the same bottom transistors, and there are totally 36 different members in *Order_SET*. The reason of using (*a b c d, e f h g*) as a representative in the above example can be explained as follows. If there exists an input pattern to generate assignment (*a b c d, e f h g*) = (1110, 1110), then all other orders in *Order_SET* can also receive the same assignment. On the contrary, if there exists no assignment with (*a b c d, e f h g*) = (1110, 1110), then it is not possible for all other members in *Order_SET* to have this assignment. In fact, these 36 orders can be treated as a single group, and any one of them can be a representative for this group as far as transistor ordering is concerned.

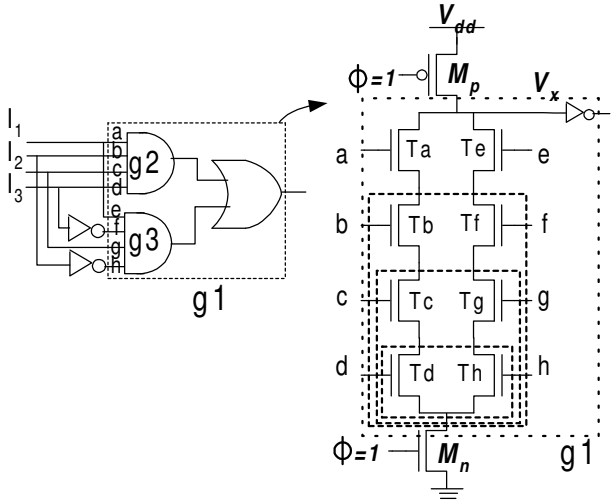

Figure 6. Transistors reordering by several iterations.

To generate all representatives for 6 CS-transistors, each transistor must be swapped to the bottom of its column. There are 16 (=4*4) representatives of different orderings at iteration-0 as shown in Figure 7. For each representative order, we need to check whether the assignment of (1110, 1110) can be generated. If there is such an assignment, then all other orders in the same group have 6 CS-transistors and will not be considered in the next iteration; otherwise, all orders in the same group has less than 6 CS-transistors and will be further considered in the next iteration.

In the next iteration (iteration-1), we only consider those groups whose representative orders have less than 6 CS-transistors in the 0^{th} iteration. If there exists no representative order to accomplish the goal of smaller than 6 CS-transistors in the 0^{th} iteration, then the process terminates and concludes that transistor reordering cannot further reduce the number of CS-transistors for the domino gate. In the above example, there is no test vector for representative order *(a b c d, e f h g)* with 6 CS-transistors, so it can be further considered for transistor reordering. In iteration-1, for each candidate group generated in iteration-0, we further divide the candidate group into subgroups by swapping transistors to the positions next to the bottom transistors as shown in Figure 7. Again, each subgroup has a representative order. For the same example, the group with representative (*a b c d, e f h g*) is divided into 9 subgroups, and each subgroup again has a representative order (Figure 7). For the representative order of each subgroup, we need to check whether it is possible to allow 5-CS transistors. If a representative order cannot allow 5 CS-transistors, we will consider it in the next iteration, *i=2*. This process is repeated in a breadth-first search manner till all *k-1* iterations are exercised, or no "eligible" group of orders can be further found. Finally, we choose an order with the least number of CS-transistors.

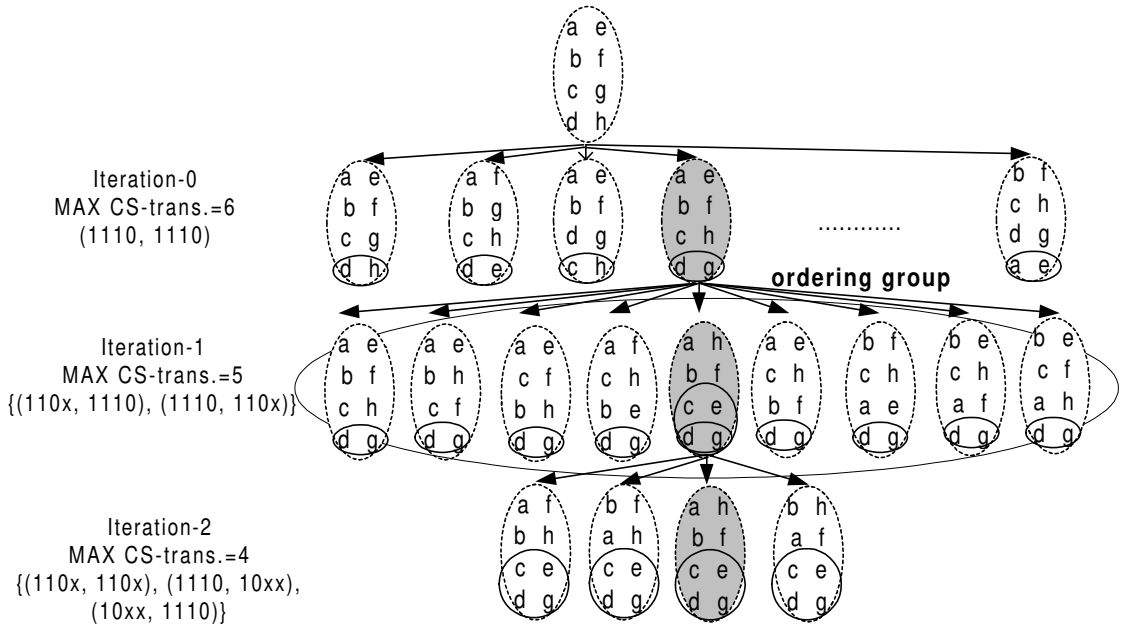

Figure 7. The transistor reordering hierarchy for the circuit in Figure 6.

3.2 A Probability-Based Heuristic for Transistors Reordering

For many circuits, the previous structural approach can exhaustively compute an optimal order with the least number of CS-transistors. However, there may exist some circuits whose execution times are very large. In this subsection, we propose a probability-based heuristic method to efficiently find a good transistor ordering. Instead of considering all representative orders, in each iteration, our heuristic method adopts a depth-first search approach. In the i^{th} iteration, our goal is to find a representative order that will **not** generate *k-i* CS-transistors. The method will first rank all representative orders by a cost function (explained later), and then process the one with the lowest cost. If the representative order with the lowest cost can generate *k-i* CS-transistors, then the heuristic will abandon it and will search the representative order with the second lowest cost. The search process is repeated until a representative order with CS-transistors smaller than *k-i* is found. If no representative order can be found with CS-transistor smaller than *k-i*, then the search process terminates; otherwise, it will go to the next iteration for considering *k-i-1* CS-transistors. In this way, the exhaustive search of transistor reordering can be avoided.

The cost function to rank representative orders is based on the following observation. Consider the 0^{th} iteration for the example in Figure 7. The objective of our heuristic is to find a representative order that is **not** likely to generate 6 CS-transistors. In other words, for the order (*a b c d, e f h g*), we would like to know the possibility of generating an assignment (*a b c d, e f h g*) = (1110, 1110). Though it is difficult to find the exact probability of assignment (*a b c d, e f h g*) = (1110, 1110), it is still possible to estimate the probability using a COP-like approach [2]. Firstly, for each domino gate, we compute its *1-controllability* using simple rules. By definition, the 1-controllability of a gate is the probability that the gate will generate the logic 1 value under random input patterns. For example, the 1-controllability of an AND gate equals to the product of the 1-controllabilities of all its inputs. Rules for the 1-controllabilities of other gate types can be derived similarly [2]. Thus, the 1-controllabilities of all domino gates in the circuit can be generated by one sweep of rule applications from primary inputs to primary outputs. Note that the probability values derived are just approximate because of signal dependencies.

Let the 1-controllability of node *n* be *p(n)*. After computing the 1-controllabilities for all nodes, we can estimate the probability of assignment (*a b c d, e f h g*) = (1110, 1110) by *p(a)*p(b)*p(c)*(1-p(d))*p(e)*p(f)*p(h)*(1-p(g))*. The process can be repeated to assign each representative order with a probability estimation. A representative order with low probability estimation for an assignment indicates that the transistor order is difficult to achieve the assignment, and should be processed first.

In Figure 7, assume we have $p(a)=p(b)=p(c)=p(d)=0.5$, $p(e)=p(f)=p(g)=0.6$, and $p(h)=0.4$. Consider two orders (*a b d c, e f g h*) and (*a b c d, e f h g*) in iteration-0 for assignment (1110, 1110). Based on the above probabilities, order (*a b d c, e f g h*) will be assigned the probability of $p(a)*p(b)*p(d)*(1-p(c))*p(e)*p(f)*p(g)*(1-p(h))=0.0081$, while order (*a b c d, e f h g*) will be assigned the probability of 0.0036. Therefore, our algorithm will choose to process order (*a b c d, e f h g*) first, because the assignment is less likely to be generated than the order of (*a b d c, e f g h*).

3.3 Timing Consideration

Since transistor ordering may also affect the speed of a domino gate, in this subsection, we discuss how to take the timing problem into consideration so that our reordering will not violate the timing constraint. In domino logic optimization, one important step is to reorder the transistors so as to speed up the propagation delay [11]. Normally, the latest arrival input is placed in the topmost position of the transistor column. In this way, all other transistors are turned on earlier than the topmost transistor, so that the source capacitance of the topmost transistor is conducted to the ground before its input (latest) signal arrives.

It is very easy to put such a requirement into our formulation. In other words, we just need to fix the latest arrival signal in the topmost position of the transistor column, while only allow other signals to be reordered for reducing the CS purpose. Since there are many orderings for the CS problem, even with the topmost transistors excluded, we still have enough searching space to choose a good ordering with the delay constraint satisfied. In addition, we emphasize that reducing the CS problem can also improve circuit area/speed due to the alleviation of the CS problem; i.e., the smaller size of the weak pull-up P-transistor leads to faster speed. Consider an example with two orderings of (*a b c d, h f e g*) and (*a b c d, e h f g*) shown in Figures 8(a) and 8(b), respectively. Each of both orderings has 4 CS-transistors. The ordering of (*a b c d, h f e g*) has less delay since both later arrival signals from inverters are placed on the topmost two positions.

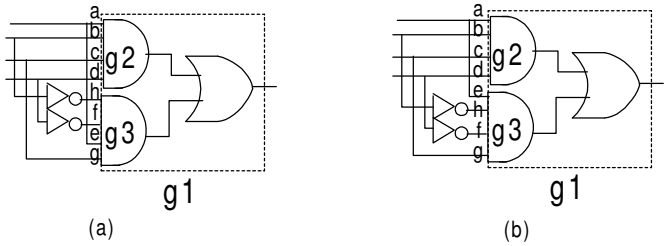

Figure 8. Transistor reordering with timing considered.

4. Fault Model and Test Generation

Test pattern generation plays a key role in this work, since it is used to find both the largest number of CS-transistors and the corresponding test vector for each domino gate. To determine the largest number of CS-transistors for a domino gate, an input assignment is given to the gate, and then the test pattern generation process is applied to determine whether a test vector can be found to achieve the assignment. As shown in Figure 5, the ATPG process is guided by a CS-state tree which gives the input assignment for each domino gate starting from the structural worst case. This section describes the relationship between CS faults and stuck-at faults. By modifying the circuit structure, the CS fault test generation can be accomplished using traditional single stuck-at fault test generation methods. Further, we will show that any set of multiple CS faults in a domino circuit can be completely detected by a set of single CS fault test patterns.

4.1 Single CS Fault

This subsection concentrates on the behavior for a single CS fault. Thus unless otherwise stated, by CS (stuck-at) fault, we mean single CS (stuck-at) fault. A CS fault behaves as if a stuck-at-1 fault appears on the output stage during the evaluation phase as shown in Figure 3, but the fault may not occur constantly depending on the amount of parasitic capacitance. In Figure 3, if only the lowest inputs on both pull-down paths are low and other inputs are high, then the probability of a CS fault occurrence is high. However, the fault might not be activated by other inputs if the charge sharing effect is too senseless to be observed. Thus, the CS fault may or may not occur depending on the input patterns applied. Unless the most sensitive test vectors is used, detection of the CS fault cannot be guaranteed.

Consequently, the detection of CS faults is different from that of stuck-at faults, because test vector generation for the former case is more restrictive. In fact, the most sensitive test pattern is the one that can turn on the maximum number of CS-transistors for the domino gate under test. Thus, to derive the most sensitive test pattern for the CS fault of a domino gate is also to derive the maximum number of CS-transistors for the gate at the same time.

4.2 Multiple CS Faults

The multiple CS fault (*MCSF*) model assumes that several single CS faults (*SCSFs*) occur simultaneously as shown in Figure 9. It is very interesting to investigate the relationship between the test vectors of *MCSFs* and the test vectors

derived based on *SCSFs*. In the case of tree circuits, there exists a set of single stuck-at fault test vectors which detect all multiple stuck-at faults [5]. Further, the set of test vectors can be derived with computing complexity $O(n^2)$ where n is the number of gates in the tree-structured combinational circuit [5]. Unfortunately, this good property does not hold in the case of non-tree-structure combinational circuits because of fault masking [5]. That is, test vectors derived based on single stuck-at fault assumption for general combinational circuits cannot guarantee the detection of all multiple stuck-at faults.

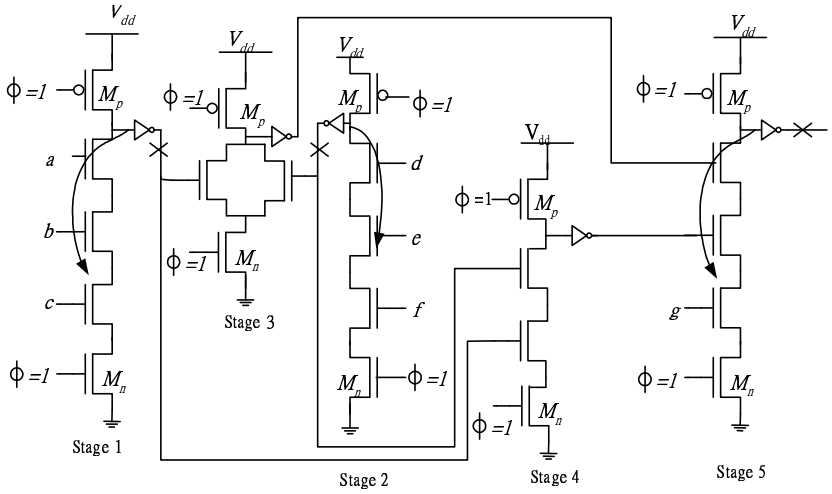

Figure 9. Multiple CS faults at stages 1, 2 and 5.

For example, the *MCSF* shown in Figure 9 contains three *SCSFs* and can be represented by the equivalent gate-level circuit given in Figure 10. Each of the *SCSF* can be detected by test vectors {*a=1, b=1, c=0, d=1, e=1, f=1, g=1*}, {*a=1, b=1, c=1, d=1, e=1, f=0, g=1*}, and {*a=1, b=1, c=1, d=1, e=1, f=1, g=0*}, respectively. It can be found that any one of these test vectors can also detect the *MCSF* in this example. Since a pure domino-logic does not contain inversion function and since any *SCSF* is functionally equivalent to a single stuck-at-1 fault, we will show that these two properties allow us to detect each *MCSF* using a set of *SCSF* test vectors for domino circuits.

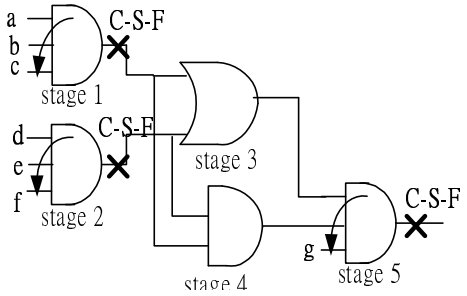

Figure 10. The gate level model for the example in Figure 9.

Let t_i be a test vector for a *SCSF* f_i fault, we have the following theorem.

Theorem 1*: In an irredundant pure-domino logic, an MCSF containing SCSFs f_1-f_n can be detected by any one of test vectors t_1-t_n.*

Proof**:** The proof can be done by induction. For easiness of discussions, the following notations will be used.

$d_1, d_2, d_3 \ldots d_n$ (the domino gate, either AND or OR gate);

$f_1, f_2, f_3 \ldots f_n$ (the *SCSFs* occurred on $d_1, d_2, d_3 \ldots d_n$);

$t_1, t_2, t_3 \ldots t_n$ (the *SCSFs* test vectors for $f_1, f_2, f_3 \ldots f_n$);

a/b (the output value of a specific line with/without an *SCSF*);

Basis case: The *MCSF* under consideration contains only two *SCSFs* f_1 and f_2 on d_1 and d_2. The discussion can be divided into three different cases.

(1) The fault effects of f_1 and f_2 do not interact.
In this case, of course, both t_1 and t_2 will detect the *MCSF*.

(2) The fault effects of f_1 and f_2 are propagated and merged at an AND (OR) gate. Since t_1 can propagate the fault effect through the AND (OR) gate, all side inputs of the AND (OR) gate must have logic value 1 (0) as shown in Figure 11(a) (Figure 11(b)). Note that the side inputs of the AND (OR) gate are inputs of the AND (OR) gate that do not propagate the fault effect of f_1. Now, under test vector t_1, fault f_2 may or may not be activated. If it is not activated, then the case is similar to the *SCSF* assumption. However, if f_2 is activated, it has logic 1 (0/1) propagated to the AND (OR) gate because CS fault always gives stuck-at-1 fault effect. Thus, the *MCSF* containing *SCSFs* f_1 and f_2 will be detected by t_1 regardless of the activation or deactivation of f_2 by t_1. Similarly, the *MCSF* can always be detected by t_2 regardless of the activation or deactivation of f_1 by t_2.

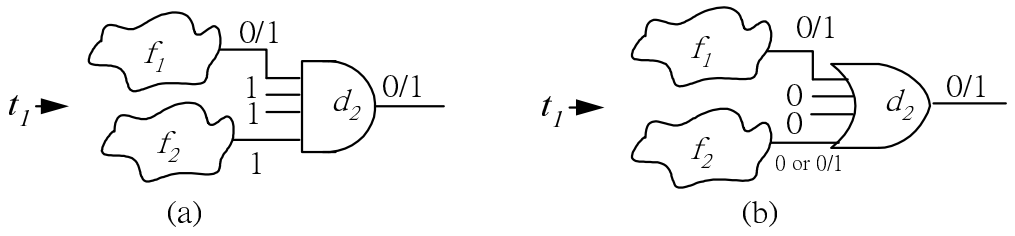

Figure 11. *MCSF* detection with two *SCSFs* (case2).

(3) The fault effect of f_1 (f_2) is propagated to d_2 (d_1).

Without loss of generality, we assume that d_1 feeds d_2. If d_2 is an AND (OR) gate, under t_1, the fault effect 0/1 of f_1 will be propagated to d_2 and the side inputs of d_2 will have logic 1 (0). Now, if f_2 also occurs, the fault effect on the output of d_2 is still 0/1 as shown in Figure 12(a) (Figure 12(b)). Thus, the fault effect of f_1 is not masked by f_2, and the *MCSF* containing f_1 and f_2 will be detected by t_1. Similarly, we can prove that t_2 also detect the *MCSF*.

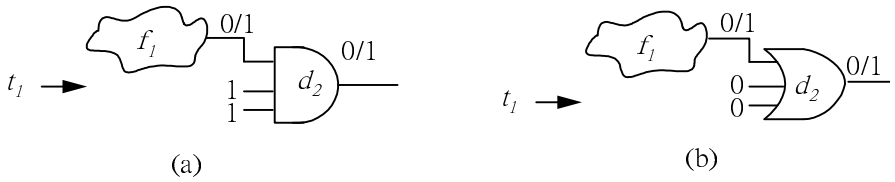

Figure 12. *MCSF* detection with two *SCSFs* (case3).

Inductive step: We will prove that if a *MCSF* containing n-1 faults, f_1, f_2, ..., f_{n-1} can be detected by any one of test vectors t_1, t_2, ..., t_{n-1}, then a *MCSF* containing n faults, f_1 - f_n, can also be detected by any one of t_1 - t_n. The proof of induction can be done similarly to the basic case, so only one case needs to be given.

The sub-circuit S_1 containing faults f_1 - f_{n-1} is parallel to the sub-circuit S_2 containing f_n , and both sub-circuits are merged at an AND (OR) gate. Based on the induction hypothesis, faults f_1 - f_{n-1}in S_1 can be detected by any one of test vectors t_1 - t_{n-1} (say t_i) and a 0/1 fault effect will be propagated from S_1 to the AND (OR) gate. Further, under the same test vector t_i, all side inputs of the AND (OR) gate have logic 1 (0) as shown in Figure 13(a) (Figure 13(b)). Now, if t_i can not activate f_n, then the case of *n*-fault detection has been done. However, if t_i activates f_n then S_2 gives a logic 1 (0/1) to the AND (OR) gate and the multiple fault containing faults f_1 - f_n still can be detected. Similarly, we can easily prove that t_n can detect the *MCSF* containing f_1 - f_n. Thus, the case has been successfully proved. **Q.E.D.**

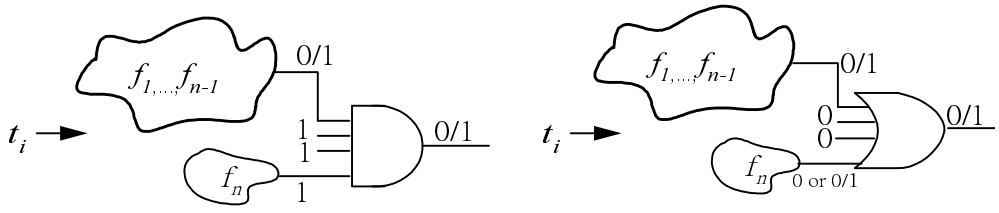

Figure 13. *MCSF* detection with n *SCSFs*.

Based on the above discussion, it can be observed that the detection of any *MCSF* is much easier than for *SCSF* in a pure domino-logic. The reason comes from the fact that an *MCSF* containing ***n*** *SCSFs* can be detected by at least ***n*** test vectors for *SCSFs* (Theorem 1). The situation is even more optimistic when more *SCSFs* are included in an *MCSF*. We have also found that no redundant CS fault exists since no CS fault masking will occur in any case as shown above. However, a pure domino-logic imposes a fundamental constraint for implementing logic functions. Traditionally, this has been dealt with by propagating intermediate inverters either backward to the primary inputs or forward to the primary outputs without increasing the number of logic levels [8], [9]. Removal of intermediate inverters will introduce hardware overhead that can be significantly reduced by an optimal output phase assignment. It can be easily verified that an *MCSF* with primary inputs or primary outputs with inverters still can be detected by any one of its corresponding *SCSF* test vectors. Thus, we have the following corollary:

Corollary*:* In an irredundant domino logic with inverters appearing only at either primary inputs or primary outputs, an *MCSF* containing *SCSFs* faults f_1 - f_n can be detected by any one of test vectors t_1 - t_n where t_i is the test vector for *SCSF* f_i.
Proof: The proof is trivial and is thus omitted.

4.3 Pseudo Gates and ATPG

In order to find the largest number of CS-transistors for a domino gate, we must find a test vector that turns on the largest number of CS-transistors. The test vector can then be used to test the (worst) CS fault of the gate. One way of achieving the goal is to modify a test generation algorithm, but this may require intensive programming work. In our algorithm, we apply a pseudo gate approach that changes the circuit structure so that the traditional ATPG for single stuck-at faults can be utilized for each CS fault. The pseudo gate approach is described as follows: For the circuit in Figure 14 (which is also in Figure 5), suppose we want to find a test vector for the structural worst case. The test vector is to turn on transistors {*Ta, Tb, Tc, Te, Tf, Tg*} and turn off transistors {*Td, Th*}. We can modify the circuit as in Fig. 14(c) and use the traditional single stuck-at fault model to obtain a test vector for the CS fault.

In Figure 14, for wire $Y \rightarrow Z$ stuck-at-1 fault, node *Y* must have the assignment 0 which can result in two assignments {*g2=0, g3=0*} in both of its inputs. Because of {*g2=0, g3=0*}, the assignments of {*a=1, b=1, c=1, d=0, e=1, f=1, g=1, h=0*} must be satisfied. The conditions to satisfy these assignments are the same as the assignments to turn on transistors {*Ta, Tb, Tc, Te, Tf, Tg*} and to turn off transistors

{*Td, Th*} for the CS fault. As shown in Figure 5(c), the structural worst case assignment cannot be satisfied. Thus, assignments {*a=1, b=1, c=0, d=x, e=1, f=1, g=1, h=0*} and {*a=1, b=1, c=1, d=0, e=1, f=1, g=0, h=x*} must be tried. This can be easily accomplished by modifying the pseudo gate shown in Figure 14(c), such that a traditional single stuck-at test generation process can deal with the new assignment. Note that any input assigned ***x*** can be removed from the pseudo gate. For example, for assignment {*a=1, b=1, c=0, d=x, e=1, f=1, g=1, h=0*}, the pseudo gate can be constructed by removing the inverter of input ***c*** and by removing input ***d***.

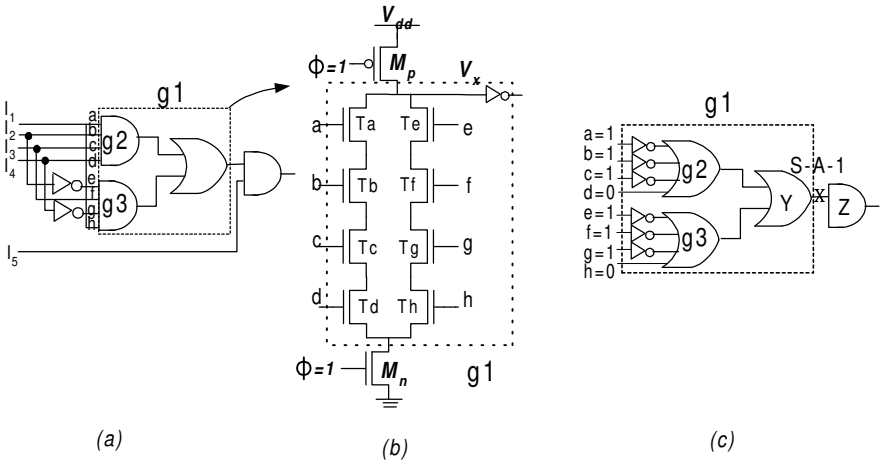

Figure 14. The example of generating a pseudo gate.

4.4 Fault Simulation and Test Length Reduction

We now discuss how the test vector length can be reduced using fault simulation. For each domino gate, one can find a test vector to activate the largest number of CS-transistors. On the other hand, there may be many of such test vectors for a domino gate. It is possible that, to activate the largest number of CS-transistors, several domino gates can share the same test vector. To reduce the number of test vectors, our algorithm makes use of a fault simulator. The algorithm first processes each domino gate in sequence. For a domino gate, if a test vector ***v*** has been found, we then perform fault simulation for test vector *v*. Suppose $\boldsymbol{n_i}$ CS transistors of another domino gate $\boldsymbol{d_i}$ can be activated by test vector *v*. The number n_i and the test vector *v* are both stored. Later on, when processing d_i, suppose the maximum number of CS-transistors activated is the same as n_i and the test generation algorithm returns another test vector v_i. Since either one of test vectors *v* and v_i can activate n_i CS-transistors, rather than using vector v_i, we can reuse vector *v*. In this way, we can save one test vector. We have implemented the ATPG algorithm in Figure 15.

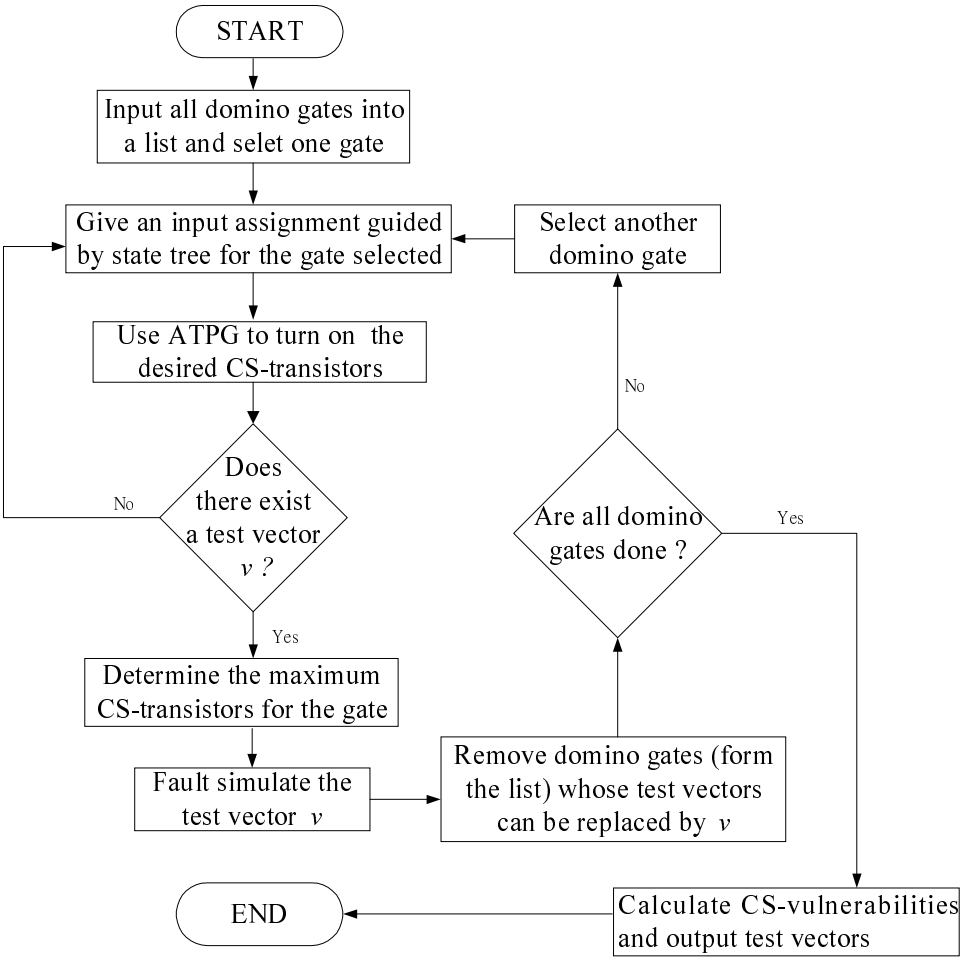

Figure 15. The flowchart of ATPG with fault simulation.

5. Experiment Results

In this section, we first present the results for CS-vulnerability derivation and test pattern generation processes for each benchmark circuit, then we show the results of CS-vulnerability improvement using the transistor reordering technique. The CS-vulnerability ($=C_{imax}/C_{ithresh}$) of a domino gate is computed as follows. First, we assume that there is no residual charge left as mentioned in Section 2. Then, the value of $C_{ithresh}$ ($=V_{dd}*C_o/V_{thresh} - C_o$) is obtained by taking an eight-input domino AO gate (e.g. ***g1*** in Figure 5) as an example (for simplicity of demonstration) to estimate the value of C_o based on a 0.35-μm TSMC CMOS process. After the design and layout processes, we extracted C_o=13.8795fF ($13.8795\times10^{-15}F$). By assuming V_{thresh}=1/2 V_{dd}, we can have $C_{ithresh}$=13.8795fF. The value of C_{imax} is determined by (the number of CS-transistors)*$C_{i\text{-}tr}$ where $C_{i\text{-}tr}$ is the source-drain parasitic capacitance for a turn-on N-transistor. We use the same eight-input domino AO gate to extract $C_{i\text{-}tr}$ as 4.42fF. Thus, we finally have C_{imax} = (the number of CS-transistors)* 4.42fF.

Table 2 shows the results of CS-vulnerability derivation and test pattern generation for each benchmark circuit. Each circuit is processed by SIS's script, script.algebraic [13] for area optimization; inverters are then pushed to primary inputs to generate non-inverted logic circuits except at primary inputs. Finally, all redundancies in each circuit are removed by the SIS command "redundancy_removal". Column two shows the total number of gates for each original random circuit, and column three shows the number of domino gates for each circuit. Note that without the fault simulation process, the number of test vectors is equal to the number of domino gates. Column four gives the total number of transistors in the structural worst case, and column five gives the total number of CS-transistors which can be activated. Column six provides the CS-transistors turn-on ratio which is obtained by dividing the value of column five by that of column four. Column seven (eight) presents the highest (average) CS-vulnerability value for each circuit. Column nine shows the ratio of vulnerable domino gates which are the gates with CS-vulnerability values larger than one. Finally, column ten gives the number of required test patterns with fault simulation executed.

For example, in circuit *C1355* of Table 2, the total number of structural worse-case transistors is 456, and the total number of CS-transistors that can be activated is 388. Thus, the CS-transistor turn-on ratio is 0.851. The largest CS-vulnerability is 1.274, while the average CS-vulnerability is 0.65. Further, out of 190 gates, there are 32 gate whose CS-vulnerability is larger than 1. So, the ratio of vulnerable gates is 16.8%. Finally, after test vector reduction, this circuit requires only 144 (instead of 190) vectors to test all CS faults. The important implications of such results for circuit designers are two fold. First, the CS problem of domino circuits is very significant (because of the large CS-vulnerability values), so it must be well solved. Second, the large variation of CS-vulnerability values in each circuit indicate that, instead of adding a weak pull-up transistors to every domino gate, circuit designers should only deal with those gates with higher CS-vulnerability values.

Table 3 gives the results of the improvement in CS-vulnerability values after transistor reordering. Similar to Table 2, the first column shows the name of each circuit, while the second column shows the number of domino gates. The rest of columns are partitioned into three portions: the initial order, the heuristic order, and the optimal order. For each domino gate in a circuit, the initial order of transistors is assumed the order of each gate that is specified in its BLIF format. The heuristic-order result is obtained using the algorithm described in Subsection 3.2, while the optimal-order result is obtained from the branch-and-bound algorithm discussed in

Subsection 3.1. For each ordering, we compute the following three different values for the CS problem. First, for each domino gate, we obtain the worst-case number (*N1*) of CS-transistors based only on circuit structure (without considering circuit function) and the maximum number (*N2*) of CS-transistors using ATPG (by considering circuit function). For each circuit, we sum up both numbers individually (called *sum(N1)* and *sum(N2),* respectively) for all domino gates. The 3th, 6th, and 9th columns titled "Turn-on ratio" show the ratios between *sum*(*N1)* and *sum*(*N2)* for each circuit, with different transistor orderings. For example, circuit *C5315* has the ratio of 0.886 for the initial order, the ratio of 0.673 for the heuristic order, and 0.671 for the optimal order. It can be found that the CS problem for circuit *C5315* has been greatly alleviated. In fact, after transistor reordering, we can greatly reduce the numbers of maximum CS-transistors (considering circuit function) for most domino gates, and this significantly relieves the CS problem for domino circuits.

For each domino gate, we also compute its CS-vulnerability using the formula of $C_{imax}/C_{ithresh}$. The average CS-vulnerability of each domino circuit is computed by summing the CS-vulnerabilities of all gates, and then the total number of domino gates in the circuit divides the summed value. The 4th, 7th, and 10th columns titled with "Aver-vul." show the average values for different orders. For example, circuit *C5315* has the average CS-vulnerability of 0.544 in the initial order, 0.413 in the heuristic order, and 0.412 in the optimal order. The 5th, 8th and 11th columns titled with "CPU (sec)" show the CPU run-time for each circuit. All experiments were conducted on ULTRA 2 machines. Several experiments for the optimal ordering cannot be completed because of large CPU time and are marked as (N/A). By comparing the results between the initial and heuristic orders, we have found that the average CS-vulnerabilities are greatly reduced in most benchmark circuits.

As mentioned before, if the CS-vulnerability of a domino gate is larger than or close to 1, then the domino gate needs special attention during the design process since the CS problem can be severe. We summarize this information in Table 4. Similar to Table 3, the first column shows the name of each circuit, the second column shows the number of domino gates, and the rests present the CS-vulnerability information for the initial order, the heuristic order and the optimal order. For example, circuit *C5315* has 35 (106) gates with CS-vulnerabilities no less than 1 (0.9) in the initial order. After the heuristic reordering process, the number of gates is reduced to 9 (30) for CS-vulnerability no less than 1 (0.9). The optimal ordering result is 7(23) gates with CS-vulnerabilities no less than 1 (0.9). By comparing the results between the initial and heuristic orders for all circuits, we have 45.2% (47.6%) of improvement

in the number of gates whose CS vulnerabilities no less than 1(0.9). Finally, we have found that the heuristic ordering can obtain fairly good results using much less CPU time.

6. Conclusion

The CS problem is a notable issue in domino logic design. Although there are methods to alleviate this problem, often, they introduce area and delay overhead. In this paper, we have introduced the concept of CS-vulnerability, which can be used to evaluate the sensitivity of the CS problem for each domino gate. A designer can make use of this information to take special care of a domino gate with high probability to have the CS problem. In addition, we have proposed an efficient transistor-reordering algorithm to minimize the CS-vulnerability for each domino gate. Our results show that the CS-vulnerabilities can be greatly reduced in many circuits. A side product of deriving the CS-vulnerability for each domino gate is the generation of test vectors for the CS faults. We have also proved that a set of test vectors for single CS faults can also detect all multiple CS faults, and this good property significantly guarantees the test quality as well.

References

[1] K. Bernstein, J. Ellis-Monaghan, and E. Nowak, " High-Speed Design Styles Leverage IBM Technology Prowess," *IBM Micro News*, Vol. 4, Number 3, 1998.

[2] F. Brglez, "On Testability of Combinational Networks," *Proc. of International Symposium on Circuits and Systems*, pp. 221-225, 1984.

[3] P. E. Gronowski, W. J. Bowhill, D. R. Donchin, R. P. Blake-Campos, D. A. Cralson, E. R. Equi, B. J. Loughlin, S. Mehta, R. O. Muller, A. Olesin, D. J. W. Noorlag, R. P. Preston, "A 433-MHz 64-b Quad-Issue RISC Microprocessor," *IEEE Transcation on Solid-State Circuits*, Vol. 31, pp. 1687-1695, Nov. 1996.

[4] P. E. Gronowski, W. J. Bowhill, R. P. Preston, M. K Gowan and R. L. Allmon, "High-Performance Microprocessor Design," *IEEE Transication on Solid-State Circuits*, Vol. 33, pp. 676-686, May 1998.

[5] W. B. Jone, P. H. Madden, "Multiple Fault Testing Using Minimal Single Fault Test Set for Fanout-Free Circuits," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, pp. 149-157, Jan. 1993.

[6] S. M. Kang, Y. Leblebici, *CMOS Digital Integrated Circuits: Analysis and Design*, McGraw-Hill Book Co., 1996.

[7] K. J. Lee, M. A. Breuer, "On the Charging Sharing Problem in CMOS Stuck-Open Fault Testing", *Proc. International Test Conference*, pp. 417-426, 1990.

[8] R. Puri, A. Bjorksten, T. E. Rosser, "Logic Optimization by Output Phase Assignment in Dynamic Logic Synthesis," *Proc. IEEE International Conference on Computer-Aidded Design*, pp. 2-8, 1996.

[9] M. R. Prasad, D. Kirkpatrick, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "Domino Logic Synthesis and Technology Mapping," *Proc. IEEE/ACM International Workshop on Logic Synthesis*, 1997.

[10] J. A. Pretorius, A. S. Shubat, C. A. Salama, "Charge Redistribution and Noise Margins in Domino CMOS logic," *IEEE Transactions on Circuits and Systems*, Vol. CAS-33, pp. 786-793, Aug. 1986.

[11] S. C. Prasad, K. Roy, “Transistor Reordering for Power Minimization Under Delay Constraint,” *ACM Transaction on Design Automation of Electronic Systems*, Vol. 1, No. 2, pp. 280-300, April 1996.

[12] P. Srivastava, A. Pua, L. Weich, “Issues in the Design of Domino Logic Circuits,” *Proc. IEEE Great Lakes Symposium on VLSI*, pp. 109-113, 1998.

[13] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Muragi, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-Vincentelli, “SIS: A System for Sequential Synthesis,” U. C. Berkeley, *Technical Report* UCB/ERL M92/41, May 1992.

[14] N. H. E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design-A Systems Perspective*, 2nd Edition, Addison-Wesley Pub. Co., 1992.

[15] Z. Wang, G. A. Jullien, W. C. Miller, J. Wang, S. S. Bizzan, “Fast Adder Using Enhanced Multiple-Output Domino Logic,” *IEEE Transaction on Solid-State Circuits*, Vol. 32, pp. 206-214, Feb. 1997.

Table 2. The results for CS-vulnerability derivation and test pattern generation.

Bench-mark circuits	Total gates count	Domino gates count	The total # of transistors for structural worst case	The total # of transistors which can be activated.	The CS-transistor turn-on ratio	The highest CS-vulnerab-ility	The average CS-vulnerab-ility	The ratio of vulner. gates	# of test vectors after fault simulation
C1355	776	190	456	388	0.851	1.274	0.650	0.168	144
C1908	564	161	322	288	0.894	1.911	0.570	0.081	125
C2670	974	261	615	545	0.886	1.911	0.665	0.046	204
C3540	1657	481	1117	968	0.867	4.458	0.641	0.083	242
C5315	2393	704	1357	1203	0.886	2.229	0.544	0.05	399
C6288	4607	1599	3128	2948	0.942	0.955	0.587	0	458
C7552	3365	1055	2083	1649	0.789	2.229	0.496	0.053	673
alu2	364	102	236	199	0.843	1.911	0.621	0.098	69
alu4	765	390	450	368	0.868	1.592	0.772	0.046	136
apex5	1102	291	812	731	0.900	2.866	0.800	0.148	161
apex6	587	183	370	351	0.949	2.229	0.611	0.066	55
b12	104	20	41	39	0.951	1.274	0.621	0.05	12
cht	120	36	74	73	0.986	0.955	0.646	0	2
comp	167	94	85	57	0.877	0.955	2.178	0.723	38
cps	1007	265	744	641	0.862	2.229	0.770	0.151	132
dalu	827	247	493	405	0.822	1.274	0.522	0.012	124
des	3933	1180	2334	2067	0.886	1.592	0.558	0.022	736
duke2	329	91	378	329	0.870	2.866	1.151	0.429	55
frg1	87	26	66	57	0.864	1.592	0.698	0.192	20
i10	2476	671	1490	1301	0.873	5.414	0.617	0.079	424
i6	312	67	239	239	1.000	1.274	1.136	0.567	40
i7	445	138	300	270	0.900	0.955	0.623	0	70
i8	797	261	634	632	0.997	2.229	0.771	0.065	93
i9	491	187	493	488	0.990	1.592	0.831	0.305	117
lal	75	21	37	31	0.838	0.955	0.470	0	18
pcler8	67	15	38	32	0.842	0.955	0.679	0	11
rot	581	162	351	325	0.926	2.548	0.639	0.093	112
sct	53	15	33	27	0.818	1.274	0.573	0.133	10
t481	574	146	374	297	0.794	1.592	0.648	0.062	105
x2	28	7	22	19	0.864	1.911	0.864	0.286	6
x4	261	60	159	148	0.931	1.592	0.786	0.067	23

N/A: Not Available

Table 3. Improvements of CS-vulnerability values after transistor reordering.

Bench-mark circuits	Domino gates count	Initial order			Heuristic order			Optimal order		
		Turn-on ratio	Aver-vul.	CPU (sec)	Turn-on ratio	Aver-vul.	CPU (sec)	Turn-on ratio	Aver-vul.	CPU (sec)
C1355	190	0.851	0.650	2.0	0.719	0.550	4.5	0.658	0.503	19.2
C1908	161	0.894	0.570	2.3	0.708	0.451	8	0.693	0.441	71.7
C2670	261	0.886	0.665	13.9	0.720	0.541	28	0.709	0.532	109.5
C3540	481	0.867	0.641	64.5	0.731	0.540	203.2	N/A	N/A	N/A
C5315	704	0.886	0.544	13.5	0.673	0.413	24.5	0.671	0.412	166.7
C6288	1599	0.942	0.587	21.6	0.855	0.533	81.2	0.855	0.532	93.5
C7552	1055	0.789	0.496	33.1	0.717	0.451	164.	0.670	0.421	4930
alu2	102	0.843	0.621	3.1	0.551	0.406	6.2	0.547	0.403	196.2
alu4	390	0.868	0.772	177.8	0.354	0.315	610	N/A	N/A	N/A
apex5	291	0.900	0.800	13.6	0.626	0.556	34	0.557	0.495	2471
apex6	183	0.949	0.611	1.5	0.619	0.399	1.9	0.614	0.395	7.0
b12	20	0.951	0.621	0.06	0.634	0.414	0.2	0.610	0.398	0.3
cht	36	0.986	0.646	0.1	0.500	0.327	0.14	0.500	0.327	0.2
comp	94	0.877	2.178	7.7	0.176	0.437	39.2	N/A	N/A	N/A
cps	265	0.862	0.770	18.6	0.449	0.401	56.8	0.414	0.370	7970
dalu	247	0.822	0.522	11.8	0.604	0.384	14	0.383	0.244	27.3
des	1180	0.886	0.558	46.3	0.561	0.354	111.6	N/A	N/A	N/A
duke2	91	0.870	1.151	2.46	0.601	0.794	24.44	N/A	N/A	N/A
frg1	26	0.864	0.698	0.2	0.788	0.637	0.5	0.773	0.625	1.0
i10	671	0.873	0.617	29.3	0.735	0.520	2381	N/A	N/A	N/A
i6	67	1.000	1.136	0.4	0.444	0.504	1.4	0.444	0.504	6.8
i7	138	0.900	0.623	4.1	0.467	0.323	4.6	0.467	0.323	17.4
i8	261	0.997	0.771	8.7	0.722	0.559	59.6	N/A	N/A	N/A
i9	187	0.990	0.831	12.4	0.744	0.625	125.7	N/A	N/A	N/A
lal	21	0.838	0.470	0.04	0.730	0.409	0.04	0.703	0.394	0.1
pcler8	15	0.842	0.679	0.1	0.579	0.467	0.07	0.579	0.467	0.2
rot	162	0.926	0.639	1.0	0.832	0.574	5	0.815	0.562	7.1
sct	15	0.818	0.573	0.02	0.636	0.446	0.02	0.636	0.446	0.2
t481	146	0.794	0.648	52.1	0.444	0.362	62.6	0.425	0.347	201.6
x2	7	0.864	0.864	0.01	0.682	0.682	0.06	0.591	0.591	1.1
x4	60	0.931	0.786	0.1	0.541	0.456	0.4	0.528	0.446	0.8

N/A: Not Available

Table 4. Improvements in the number of vulnerable gates after transistor reordering.

Bench-mark circuits	Domino gates count	Initial order		Heuristic order		Optimal order	
		Vul. gates ≥1	Vul. gates ≥0.9	Vul. gates ≥1	Vul. gates ≥0.9	Vul. gates ≥1	Vul. gates ≥0.9
C1355	190	32	40	16	32	16	16
C1908	161	13	20	4	17	1	10
C2670	261	12	91	6	64	4	57
C3540	481	40	98	34	69	N/A	N/A
C5315	704	35	106	9	30	7	23
C6288	1599	0	495	0	302	0	299
C7552	1055	56	104	39	78	34	65
alu2	102	10	23	5	10	5	10
alu4	390	54	153	11	29	N/A	N/A
apex5	291	43	105	17	50	15	21
apex6	183	12	22	4	9	4	9
b12	20	1	4	0	1	0	0
cht	36	0	1	0	0	0	0
comp	94	68	68	12	21	N/A	N/A
cps	265	40	106	7	31	5	20
dalu	247	3	10	0	3	0	2
des	1180	26	129	3	29	N/A	N/A
duke2	91	39	57	20	33	N/A	N/A
frg1	26	5	7	4	7	4	6
i10	671	53	126	36	82	N/A	N/A
i6	67	38	67	0	0	0	0
i7	138	0	30	0	0	0	0
i8	261	17	107	15	27	N/A	N/A
i9	187	57	66	57	58	N/A	N/A
lal	21	0	1	0	0	0	0
pcler8	15	0	8	0	0	0	0
rot	162	15	38	10	31	9	29
sct	15	2	4	0	2	0	2
t481	146	9	36	0	9	0	4
x2	7	2	2	1	2	0	1
x4	60	4	34	0	2	0	2
Average		1	1	0.452	0.476		

N/A: Not Available