False-Noise Analysis for Domino Circuits

A. Glebov[1], S. Gavrilov[1], V. Zolotov[2], Chanhee Oh[2], R. Panda[2], M. Becer[2]
[1] Microstyle - Moscow, Russia, [2] Motorola, Inc. - Austin, TX

Abstract

High-performance digital circuits are facing increasingly severe noise problems due to cross-coupled noise injection. Traditionally, noise analysis tools use the conservative assumption that all neighbors of a net can switch simultaneously, producing the worst-case noise. However, due to logic correlations in the circuit, this worst-case noise may not be realizable, resulting in a so-called false noise failure. Some techniques for computing logic correlations have been designed targeting static CMOS circuits. However high performance microprocessors commonly use domino logic for their ALU. The domino circuits have lower noise margins than static CMOS circuits and are more sensitive to coupled noise. Any unnecessary pessimism of the noise analysis tool results in large number of false noise violations and either requires additional extensive SPICE simulations or circuit over-design. Unfortunately false noise analysis developed for static CMOS circuits [11] fails to compute many logic correlations in domino circuits. In this paper we propose a novel technique of computing logic correlations in domino circuits. It takes into account the fact that both pull up and pull down networks of a domino gate can be in non conducting state. The proposed technique generates additional logic correlations for such states of domino gates. In order to improve the capability of logic correlation derivation technique we combine the resolution method with recursive learning algorithm[12]. The proposed technique is implemented in an industrial noise analysis tool and tested on high performance ALU blocks.

1 Introduction

Noise can occur in a circuit through a number of different mechanisms - the most prominent of which is through cross-coupling capacitance. The voltage on a net is compromised due to a transition on a neighboring net with which it has capacitive coupling. The net on which noise is injected is referred to as a *victim* net, while the net that injects noise is referred to as an *aggressor* net. The combination of a victim net and all its aggressor nets is called a noise cluster. If a victim net is not switching at the time of noise injection, a noise pulse can propagate to a latch and change the state of the circuit. This noise type is referred to as a *functional noise* and can result in a functional failure of the circuit. On the other hand, if a victim net transitions at the time of noise injection, the delay of the victim transition is altered. This type of noise is referred to as *delay noise* and can result in a performance violation. Here we do not consider noise on delay, though the proposed approach can be applied to it.

Noise analysis tools typically make the assumption that all aggressor nets switch in the same direction at the worst alignment time [1], [5], [7]. Under this assumption, the noise injected from each aggressor combines, creating the maximum possible composite noise pulse on the victim net. However the timing and logic constraints present in the circuit prevent aggressors from switching in the same direction at the worst alignment time. Therefore, the noise reported by the analysis that does not account for timing and logic correlations can severely overestimate the actual noise and create a so-called *false noise violation*. Industrial noise analysis approaches have exploited timing correlations in circuits to reduce pessimism in noise analysis by identifying situations where aggressor nets cannot switch at the same time. To determine when a net can switch, the so-called *switching windows* are propagated in the circuit using static timing analysis [1], [2], [5]. After switching windows are identified for each aggressor, the possibility of overlap between the timing windows for a set of aggressors is determined. However, this approach does not identify situations when aggressor nets can switch individually, but cannot *all* switch at the same time in the same direction due to logic relationships in the circuit. Therefore, timing correlations do not remove all false noise failures, although it has been shown in practice to be relatively effective [1].

In order to identify all false noise failures, both timing and logic correlations of the circuit must be taken into account. In [2] this problem was represented as a search for a worst-case 2-vector test using a Boolean Constraint Optimization formulation. In [3], a method based on compatible observability don't care sets was proposed. In [6] a test pattern generation approach was proposed. However, all these methods have very high computational complexity and are not suitable for large problem sizes. Since noise primarily occurs in chip-level routes, it is critical to perform false noise analysis at this level in large designs, and hence heuristic methods must be employed.

In [8], an approach based on so-called simple logic implications (SLI) [4] was developed. An SLI expresses a logic relationship between pair of signals. Initially SLIs are generated for logic gates and then more implications are derived by forward and backward SLI propagation. Unfortunately SLIs cannot capture multiple signal correlations existing in a circuit. In [11] it was proposed to use the resolution method, that has been widely used in mechanical theorem proving [10]. The resolution method operates on transistor level circuits and does not require extraction of logic function. This is particularly useful for circuits with large and complex DC-connected components (DCCCs) for which it is difficult or impossible to extract logic functions. Signal correlations for DCCC are generated assuming that when a MOS transistor is in the conducting state then its source and drain are at the same potential. For static CMOS circuits this relationship is the only reliable source of signal correlations. However in domino circuits both pull up and pull down networks can be off in evaluation phase. Ignoring this results in losing many signal correlations. Another specific feature of high performance domino logic is redundant signal coding that creates lots of signal correlations. Our experience showed that the algorithm of logic constraints derivation proposed in [11] cannot compute many of such logic constraints in reasonable time.

Domino circuits make up a significant portion of high performance microprocessors and have rather low noise margins. Therefore neglecting domino specific signal correlations significantly increases pessimism of noise analysis and results in circuit over-design. In this paper, we propose a novel technique for computing logic correlations between signals in high performance domino circuits and apply this technique for false noise analysis. We developed an algorithm for generating logic correlations for domino gates that takes into account that both pull up and pull down networks can be in non conducting state. We also improved efficiency of logic constraints derivation for high performance domino circuits by combining the recursive learning algorithm[12] with the resolution method. We implemented the proposed technique in industrial noise analysis tool Clarinet [1] used in our design flow for high performance microprocessors. Our experience shows that the proposed technique is very efficient for domino circuits and helps filter out as mach as 29% of aggressor nets in noise analysis.

The proposed method uses a zero-delay assumption which is conservative only for glitch-free circuits, obtained, for instance, through special transistor sizing methods. Fortunately domino circuits are also glitch-free.

The remainder of this paper is organized as follows: Section 2 explains the concept of logic constraints and their derivation by the resolution method. Section 3 describes generation of logic constraints for domino circuits. Section 4 explains application of logic constraints to false noise analysis. Section 5 shows experimental results and compares efficiency of different false noise analysis techniques. Section 6 presents our concluding remarks.

2 Logic constraints and false noise analysis

A signal in a digital circuit is always correlated with other signals in the circuit. These correlations prohibit circuit nets from having certain combinations of signals. These constraints on signal combinations in turn prohibit certain signal transitions and consequently make certain noise injection scenarios impossible.

2.1 Logic correlations of circuit signals

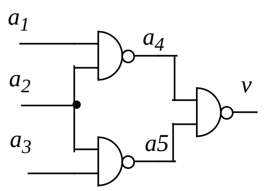
The complete set of logic correlations can be specified by listing all the prohibited signal combinations. For practical purposes it is more convenient to represent logic correlations in the form of Boolean equation $F = 0$ [11]. The signal combinations that satisfy to this equation are prohibited combinations or, in other words, logic constraints. This equation describes logic correlations in more compact form than full list of prohibited signal combinations. The most convenient form of representing function F in this equation is sum of products (SOP):

$$\sum_i \prod_j s_{i,j}^{\alpha_{i,j}} = 0 \qquad \text{(EQ 1)}$$

where s^{α} represents s if $\alpha = 0$ or its negation $\bar{s}$ if $\alpha = 1$.

Instead of writing full expression of function F in SOP form we can specify the set of its terms. Each term corresponds to a prohibited signal combination. Figure 1 shows an example of a circuit and its logic constraints for analyzing noise at net v when it is at 0 and aggressor nets are rising. A logic constraint involving only two variables is called a simple logic implication (SLI) [8]. For example, in Figure 1 SLI $\bar{v}*\bar{a}_4$ prohibits combination $v=0$, $a_4=0$. SLI are useful for deriving logic constraints by combining them with other signal correlations.

Functions specifying all the possible logic constraints are often too large and cumbersome for practical purposes. However for false noise analysis it is not necessary to use the complete set of logic constraints. Even incomplete and relatively small part of logic constraints can provide valuable information for false noise analysis. The above representation is convenient because it is valid even for incomplete sets of logic constraints.



Logic constraints for analyzing noise at net v when it is at 0 and aggressor nets are rising:

$\bar{v}*\bar{a}_4$, $\bar{v}*\bar{a}_5$,
$\bar{a}_1*\bar{a}_4$, $\bar{a}_2*\bar{a}_4$, $\bar{a}_2*\bar{a}_5$, $\bar{a}_3*\bar{a}_5$
$a_1*a_2*a_4$, $a_2*a_3*a_5$

Constraints derivation by the resolution rule:
$\bar{v}*\bar{a}_4$, $a_1*a_2*a_4$ -> $a_1*a_2*\bar{v}$
$\bar{v}*\bar{a}_5$, $a_2*a_3*a_5$ ->$a_2*a_3*\bar{v}$

Figure 1. Example of circuit and logic constraints

2.2 Sources of signals correlations

There are two possible sources of logic correlations in a digital circuit: correlation between circuit input signals and correlation due to circuit structure itself. The first type of logic correlations is related to circuit functionality and coding of its input signals. This kind of correlations can be, for example, one hot (or one cold) coding where only one signal from a given group is 1 (0). Such logic constraints need to be specified only by circuit designer as they cannot be deduced from the circuit itself. The second type of signal logic correlations follows from logic functions of circuit cells and their connections with each other. For instance, NAND gate implementing function $x=\overline{a*b}$ has three logic constraints on its output and input signals: $\bar{a}*\bar{x}$, $\bar{b}*\bar{x}$, $a*b*x$. It means that the output signal cannot be at 0 if any of the input signals is 0 and the output signal cannot be at 1 if all the inputs are at 1.

If a circuit is represented at transistor level, it is rather difficult to convert it to logic level representation. Therefore it is more convenient to use logic correlations between signals of a MOS transistor directly[11]. They are as follows:

- n MOS transistor (s - source, g - gate, d - drain) has two constraints: $g*s*\bar{d}$, $g*\bar{s}*d$
- p MOS transistor also has two constraints: $\bar{g}*s*\bar{d}$, $\bar{g}*\bar{s}*d$
- If the source of p transistor is connected to Vdd, it has constraint $\bar{g}*\bar{d}$.
- If the source of n transistor is connected to ground it has constraint $g*d$.

All the initial constraints are consequences of MOS transistor operation in its *on* state. For example, constraint $g*s*\bar{d}$ for n transistor means that if its gate is at high voltage it is impossible to keep its source at high voltage and drain at low and vice versa. The off state of MOS transistor does not imply any logic correlations because states of its drain and source are not correlated.

2.3 Derivation of logic constraints

Using an initial set of logic constraints we can apply laws of logic calculus to derive more logic correlations between signals. Moreover new logic constraints can be derived for signals that are not necessarily inputs or outputs of the same logic cell or transistor. Derivation of new logic relations from the existing ones is a well studied problem of mechanical theorem proving[10]. One of the most efficient techniques for this is the resolution method. It is based on recurring application of the resolution rule:

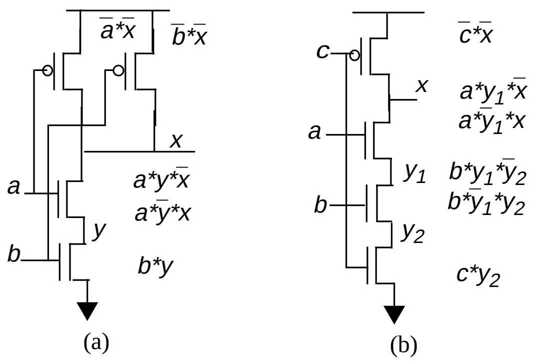
$$a \cdot B = 0, \bar{a} \cdot C = 0 \rightarrow B \cdot C = 0 \qquad \text{(EQ 2)}$$

where B and C are arbitrary logic expressions.

For false noise analysis, the resolution rule is applied to the terms of (EQ1) which specify logic constraints. Therefore *B* and *C* are products of circuit signals or their negation. Unlike the classical resolution method that derives new true sentences from known true sentences, our formulation is applied to set of false sentences (logic constraints or prohibited signal combinations) and derives new false sentences. The resolution rule fully covers all the other laws of logic calculus. Any logic consequence that can be obtained by applying any combination of logic rules can be derived by applying a sequence of resolution rules. For simplicity equality signs and boolean constant *0* are omitted from the resolution rule:

$$a \cdot B, \bar{a} \cdot C \rightarrow B \cdot C \qquad \text{(EQ 3)}$$

The resolution rule can be applied both at transistor and logic levels. At transistor level the resolution rule is applied to signals of each DC connected component (DCCC) separately. Application of the resolution rule at transistor level derives logic constraints between DCCC input and output signals. The derivation starts from set of constraints for individual transistors and continues recurrently adding new logic constraints constructed by the resolution rule from pairs of the existing ones [11]. This process continues until either it is impossible to construct new logic constraints or the allocated computational resources are exhausted. Figure 2 demonstrates the derivation of logic constraints for static and dynamic 2-input NAND gates.



Static NAND2 constraints
$\bar{a}*\bar{x}$, $\bar{b}*\bar{x}$,
$b*y$, $a*\bar{y}*x$ -> $a*b*x$

Dynamic NAND2 constraints
$\bar{c}*\bar{x}$
$c*y_2$, $b*y_1*\bar{y}_2$ -> $c*b*y_1$
$c*b*y_1$, $a*\bar{y}_1*x$ -> $c*a*b*x$

Figure 2. DCCC logic constraints calculation by resolution

Derivation of logic constraints at logic level starts from the constraints obtained for DCCCs. Since logic circuits of real digital blocks are usually very large, uncontrolled application of the resolution rule in all possible ways usually results in wasting time for generation of the same constraints multiple times. [11] proposes a heuristic approach to generate new logic constraints by combining existing simple logic implications with signal correlations for logic gates. SLIs are propagated back and forward through logic gates where they are combined with signal correlations of those gates by the resolution rule. This process continues until it is impossible to construct new logic constraints or all the allocated computational resources are exhausted. Figure 1 illustrates derivation of logic constraints on logic level.

Since logic correlations for a MOS transistor relate to its on state, the constraints derived from them describe the behavior of a CMOS gate when either its pull up or pull down network is on. The operation of a domino gate includes the situation when both the pull up and pull down networks are off. In Figure 2 we see that the derived constraints do not include constraints $\bar{a}*\bar{x}$ and $\bar{b}*\bar{x}$ for the case when both pull up and pull down networks are off and output *x* stores precharge value 1. The technique described in [11] cannot generate all the logic constraints for a domino circuit and cannot filter out a significant portion of false noise. This problem is addressed below.

3 Logic constraints for domino circuits

The principal part of a domino gate is a dynamic gate shown in Figure 2. A dynamic gate consists of its pull up and pull down networks. The pull up network consists of a P transistor controlled by the clock signal. The pull down network includes N transistors implementing a logic function in series with an N transistor controlled by the clock signal. (For simplicity we consider only footed domino logic but the proposed technique can be applied to footless gates too.) During precharge the clock signal is low, the pull up transistor is on and, the pull down network is in non conducting state due to the transistor controlled by the clock signal. The output node is precharged to high voltage. During evaluation, the P transistor is off, the transistor of the pull down network controlled by clock is on and, the state of the output node depends on the transistors implementing logic function. If these transistors create a conducting path, the output discharges to low voltage otherwise it remains at high voltage. The behaviour of a dynamic gate significantly differs from the static CMOS logic and cannot be described only by logic correlations based on the conducting state of transistors. It has additional logic correlations related to the gate behaviour when the pull down network is in non-conducting state during evaluation phase. Limiting logic correlation with only those proposed in [11] results in losing many logic constraints and consequently in noise overestimation.

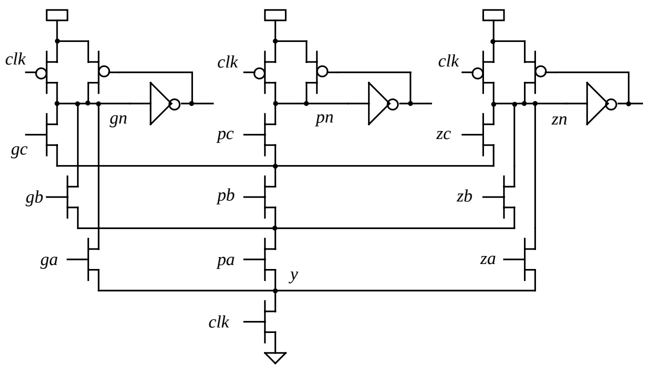


Figure 3. Complex domino gate

3.1 Transistor level logic constraints

All the constraints generated for static CMOS gates[11] are valid for domino gates too. Additionally a domino gate has logic correlations related to evaluation phase when its pull down network is off and there is no transition at the gate output. These constraints are based on the assumption that the clock signal correctly defines precharge and evaluation phases. Therefore these logic correlations are between a gate output and data (non clock) inputs only. Here we call the output of the dynamic gate as a domino gate output (*gn, pn, zn* in Figure 3). Additional set of constraints for domino gate output x includes all the possible constraints of type $\overline{g}_1*\overline{g}_2*...*\overline{g}_n*\overline{x}$, where $\{g_1,g_2,...,g_n\}$ is the minimum set of non clock inputs turning off the pull-down network of that node. In other words, applying 0 to $\{g_1,g_2,...,g_n\}$ and 1 to the other gate inputs in evaluation phase produces 1 at output x. However changing 0 to 1 at any of the inputs $\{g_1,g_2,...,g_n\}$ results in 0 at output x. The circuit shown in Figure 3 has the following additional constraints:

- $\overline{ga}*\overline{pa}*\overline{gn},\ \overline{ga}*\overline{gb}*\overline{pb}*\overline{gn},\ \overline{ga}*\overline{gb}*\overline{gc}*\overline{gn}$ for node *gn;*
- $\overline{za}*\overline{pa}*\overline{zn},\ \overline{za}*\overline{zb}*\overline{pb}*\overline{zn},\ \overline{za}*\overline{zb}*\overline{zc}*\overline{zn}$ for node *zn*
- $\overline{pa}*\overline{pn},\ \overline{pb}*\overline{pn},\ \overline{pc}*\overline{pn}$ for node *pn*

In order to develop an efficient procedure for computing additional constraints for a domino gate, we consider its pull down network as a two terminal switching network. This network consists of all the pull down transistors affecting the output of the domino gate except the transistor controlled by the clock signal. The pull down network for output *gn* of the gate shown in Figure 3 consists of transistors *ga, gb, gc, pa, pb*. Its terminals are *gn* and *y*.

For a two terminal switching network we can compute its logic function in the sum of products form:

$$F = A + B + C + ... \qquad \text{(EQ 4)}$$

where: $A=a_1*...*a_p$, $B=b_1*...*b_q$, $C=c_1*...*c_r$, ... and $a_1,...,a_p$, $b_1,...,b_q$, $c_1,...,c_r$,... are the gate inputs. This function can be computed either by traversing all the network paths or by the resolution method. The second approach is more efficient. It is as follows:

1. Compute the set *C* of logic constraints for all the network transistors.
2. Deduce new logic constraints from the set *C* by applying the resolution rule to pairs of the existing constraints. The resolution rule is applied so that to exclude variables corresponding to non terminal nodes of the network.
3. Add the deduced constraints to the set *C*.
4. Repeat steps 2 and 3 until it is not possible to generate constraints.
5. Exclude from set *C* all the terms having variables corresponding to internal nodes of the network.
6. Construct the function *F* of the switching network as sum of all the terms from set *C*.

Each term of the sum (EQ4) corresponds to a path in the switching network. Inverting this function *F* we obtain function $\overline{F}$ describing non conducting state of the network. Applying De Morgan laws we transform function $\overline{F}$ to the product of sums form:

$$\overline{F}= (\overline{a}_1+...+\overline{a}_p) * (\overline{b}_1+...+\overline{b}_q) * (\overline{c}_1+...+\overline{c}_r) *... \qquad \text{(EQ 5)}$$

Performing all multiplications and eliminating all tautologies we transform function $\overline{F}$ in the sum of products form:

$$\overline{F} = \sum \bar{a}_i \cdot \bar{b}_j \cdot \bar{c}_k \cdot \ldots \qquad \text{(EQ 6)}$$

Each term $\overline{a}_i*\overline{b}_j*\overline{c}_k*...$ of this equation defines a condition that the switching network is in non conducting state. Multiplying all these terms by inverted output signal $\overline{x}$, we obtain the required set of additional logic constraints in the form $\overline{a}_i*\overline{b}_j*\overline{c}_k*...*\overline{x}$.

3.2 Gate level constraints

The technique used for deriving logic constraints for static CMOS circuits, computes valid constraints for domino circuits as well. However high performance domino circuits often use redundant signal coding like *g (generate)*, *p (propagate*) and *z (zero*) signals in adders. Such logic correlations, once established, propagate deep into the circuit. For example, the large domino gate in Figure 3 is supposed to have correlations between its input signals like *ga*pa, pa*za, ga*za*, etc., which propagate to its outputs *g,p,z*. Unfortunately that kind of correlations are difficult to derive by the SLI propagation[11]. It requires too many forward and backward SLI passes resulting in extensive memory and computation time. The SLI propagation is efficient only for correlations following from each other according to topological order of the circuit nodes. Single pass of SLI propagation can derive many constraints but some constraints require too many passes. There exists another implication technique - recursive learning, widely used in circuit testing, optimization and verification[12]. Single application of recursive learning can compute only one constraint so it is not efficient for computing large sets of constraints. However recursive learning can easily check if an individual constraint is valid. We combine both techniques as they complement each other. SLI propagation efficiently generates many “easy constraints” and recursive learning checks validity of “difficult constraints”, which become the basis for the next pass of SLI propagation. By recursive learning we derive SLIs between gate inputs and outputs while the SLI propagation computes everything that it can.

Figure 4 shows our version of the recursive learning algorithm. Unlike the traditional approach we apply recursive learning to ex-

```
int R_learning(variables v_1,v_2,..., values x_1,x_2,...)
1. substitute variable values into existing constraints
2. if any constraint is violated (is 0)
        2.1 return 0
3. else if all the variables are assigned values
        3.1 return 1
4. select a variable w without assigned value
5. compute
      r=R_learning(variables v1,v2,..., w, values x1,x2,..., 0)
6. if r=1
        6.1 return 1
7. else
        7.1 compute
            r=R_learning(variables v1,v2,..., w, values x1,x2,..., 1)
        7.2 if r=1
                7.2.1 return 1
        7.3 else
                7.3.1 return 0
```

Figure 4. Recursive learning algorithm

isting logic constraints but not to the gate functions because our false noise analysis works at transistor level and does not extract gate functions. The input of the algorithm is a logic constraint for which we would like to check if it follows from the existing constraints. The constraint is defined by the names of its variables *v1, v2*,... and their values x_1, x_2... The algorithm returns 0 if this logic combination is prohibited or 1 if it is not. If the combination is prohibited, it is a logical consequence of the existing constraints.

Figure 5 shows the complete algorithm of computing logic constraints. First it computes constraints between pins of each logic gate (DCCC), i.e. primary logic constraints, and then derives new constraints from them. After each phase of recursive learning we perform the SLI propagation by the resolution technique. The two techniques exchange constraints that amplifies their efficiency.

```
1. Topologically sort gates.
2. Compute primary static CMOS logic constraints
3. Compute primary domino logic constraints
4. Select set of SLIs from existing constraints
5.Repeat till convergence or resources exhausted
   5.1. for each domino gate
      5.1.1 for each gate output x
         5.1.1.1 for each gate input/output a different from x
            5.1.1.1.1. r=R_learning(variables x,a, values 0,0,)
            5.1.1.1.2. if r=0
               add SLI $\overline{x}*\overline{a}$
            5.1.1.1.1. r=R_learning(variables x,a, values 0,1,)
            5.1.1.1.2. if r=0
               add SLI $\overline{x}*a$
            5.1.1.1.1. r=R_learning(variables x,a, values 1,0,)
            5.1.1.1.2. if r=0
               add SLI $x*\overline{a}$
            5.1.1.1.1. r=R_learning(variables x,a. values 1,1,)
            5.1.1.1.2. if r=0
               add SLI x*a
   5.2. generate SLIs by forward and backward propagation
```

Figure 5. Computation of logic constraints

4 False noise analysis with logic constraints

After computing all the logic constraints false noise analysis is performed separately for each noise cluster and different types of functional noise: for the victim net in either a stable low or high state, while the aggressor nets are rising or falling. Logic correlations prohibit circuit nets from having certain combinations of signals that in turns prohibits certain aggressor nets transitions. Aggressor nets $(a_1,a_2,...,a_n)$ cannot switch simultaneously in the same direction if at least one of the two signal combinations $(a_1=1,a_2=1,...,a_n=1)$ and $(a_1=0,a_2=0,..,a_n=0)$ is prohibited at the condition that the victim net is at the given state. False noise analysis searches for the subset G of aggressor nets that can switch simultaneously and inject maximum amount of noise without violating logic constraints. This noise is called maximum realizable noise. Computation of the maximum realizable noise is a combinatorial optimization problem:

$$\underset{G \subset C}{maximize} \sum_{i \in G} v_i \qquad \text{(EQ 7)}$$

subject to $R_j \not\subset G$

where:

- v_i is noise injected by aggressor net i
- G is the set of the aggressor nets that inject the combined noise
- C is a set of all the aggressor nets of the given noise cluster
- R_j is the set of aggressor nets prohibited from simultaneous switching by logic constraints

This problem is NP hard as its special case, the Maximum Weighted Independent Set problem is NP hard [11]. Fortunately the number of aggressor nets usually is not very large (about 10) and we can use an exhaustive search algorithm. Our algorithm does not construct explicitly the full list of prohibited aggressors sets. Instead we represent noise cluster constraints as a ROBDD of the Boolean function, called a characteristic function of noise cluster. The domain of the function is all combinations of the signals at victim and aggressor nets. The characteristic function equals *1* for signal combinations satisfying all the constraints and equals *0* otherwise. Construction of characteristic ROBDD is a simple recursive procedure of analysing all possible values of aggressor and victim nets and deriving possible conclusions from the logic constraints [11]. Using the characteristic ROBDD we calculate the maximum realisable noise by finding the maximum weighted set of the aggressors for which simultaneous switching of the same type is not prohibited. It is the maximum weighted set of aggressors $\{a_{i1},a_{i2},...,a_{im}\}$, for which ROBDD has two paths from its root to the terminal 1-vertex $(v=V,a_{i1}=0,a_{i2}=0,...,a_{im}=0)$ and $(v=V,a_{i1}=1,a_{i2}=1,...,a_{im}=1)$ where V is the victim state corresponding to the analysed noise. Figure 6 demonstrates the charac-

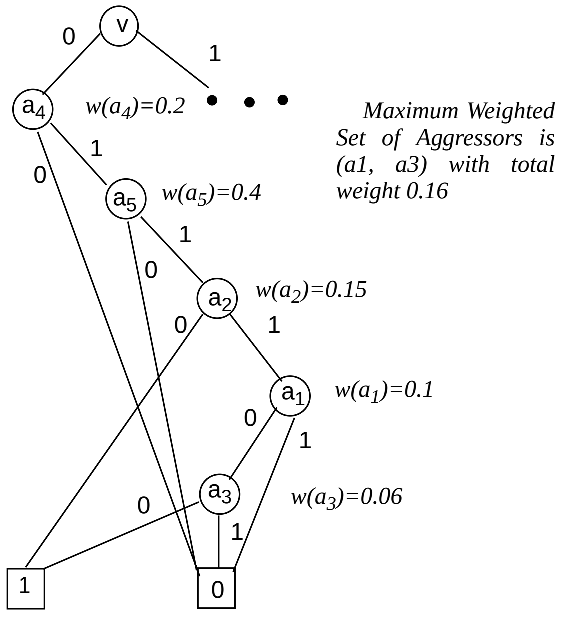

v
0
1
a4
w(a4)=0.2
Maximum Weighted Set of Aggressors is (a1, a3) with total weight 0.16
a5
w(a5)=0.4
a2
w(a2)=0.15
a1
w(a1)=0.1
a3
w(a3)=0.06
1
0

Figure 6. Characteristic ROBDD of noise cluster

teristic ROBDD for the circuit shown in Figure 1. The ROBDD describes logic constraints for analyzing the noise injected into net *v* in its low state by nets *(a1,a2,a3,a4,a5)* with rising transitions. The noise injected by each aggressor is written near the corresponding ROBDD vertices.

5 Implementation and experimental results

The proposed false noise analysis algorithm is implemented in an industrial noise analysis tool called Clarinet [1]. The system was architected using a separate false noise analysis engine called DiNo [11]. First, the noise analysis tool performs the traditional noise analysis without using logic information. It generates a list of critical noise clusters where the total noise injected into the victim is higher than the tolerable noise threshold. Each critical noise cluster is specified by its victim net and a list of aggressors with their noise values. The false noise analysis engine reads transistor level circuit description and a list of critical noise clusters. Then it generates logic constraints that could be useful for false noise analysis of at least one critical cluster. For each critical cluster we build characteristic ROBDD and find the maximum weighted aggressor set and the maximum feasible noise. The analysis can be performed both at the block and chip levels.

Initially the false analysis tool implemented the resolution method of deriving general logic constraints valid for any CMOS circuit. Our practice showed that although the technique was sufficiently good for static CMOS circuits it failed to compute many existing signal correlations in high performance domino circuits. Therefore we used additional user specified constraints to reduce the pessimism of noise analysis. Then we added the technique described in this paper. It allows us to generate all the constraints that we initially had to specify manually. The results of applying the modified false noise analysis algorithm to industrial high performance domino circuits are given in Table 1.

Circuit	#aggressors	Aggressor nets rejected by:			
		Timing windows only	User's constraints	Static and user's logic constraints	New technique
ckt1	4956	1038 / 20.9%	1621/ 32.7%	1694 / 34.2%	1796 / 36.2%
ckt2	3845	822 / 21.4%	1359 / 35.3%	1355 / 35.2%	1500 / 39.0%
ckt3	3360	618 / 18.4%	1094/ 32.6%	1385 / 41.2%	1504 / 44.8%
ckt4	4346	696 / 16.0%	1088 / 25.0%	1125 / 25.9%	1344 / 30.9%
ckt5	3008	674 / 22.4%	1324/ 44.0%	1545 / 51.4%	1557 / 51.8%

Table 1: Noise analysis results

Column 1 and 2 give the name of the circuit and the total number of aggressor nets. Columns 3 through 6 show the number and percentage of the aggressors rejected by using different types of false noise analysis:

- column 3: timing windows only;
- column 4: timing windows with user specified constraints
- column 5: timing windows combined with logic correlations of general type (initial version of false noise analysis [11]) and with user specified constraints;
- column 6: timing windows combined with the presented algorithm but without any user specified constraints.

In this table we can see that the presented technique combined with timing windows technique reduces the pessimism of noise analysis tool up to 51.8%, of which 29.4% was reduced by logic correlations. Moreover comparing columns 5 and 6 we see that the new algorithm allows complete elimination of manually specified logic constraints and achieves even better false noise reduction than user specified constraints.

6 Conclusions

In this paper, we presented a novel approach for false noise analysis of high performance domino circuits by taking into account signal correlations that prohibit aggressor nets from simultaneous switching and injecting combined noise. We developed a technique of computing signal correlations in high performance domino circuits. We achieved high efficiency of the proposed technique by correctly considering operation of domino gates when both the pull up and pull down networks are in non conducting state. We extended the resolution technique of logic constraints derivation by combining it with the recursive learning algorithm. This approach helps generate additional logic constraints for circuits with redundant signal coding that are common in high performance ALU. The proposed technique is implemented in our noise analysis tool where it is combined with false noise analysis technique for static CMOS circuits. The proposed technique significantly reduces pessimism of noise analysis tool. It eliminates the necessity for manual specification of logic constraints that otherwise were necessary for obtaining appropriate noise analysis results. The presented results of analyzing coupling noise in industrial high performance circuits demonstrate that the proposed technique reduces the number of valid aggressor nets by up to 29%.

7 References

[1] Levy, R., Blaauw, D., Braca, G., Dasgupta, A., Grinshpon, A., Chanhee Oh, Orshav, B., Sirichotiyakul, S., Zolotov, V., “ClariNet: a noise analysis tool for deep submicron design”, DAC 2000

[2] P.Chen, K.Keutzer. “Towards True Crosstalk Noise Analysis”, ICCAD-99, pp.132-137.

[3] D.A.Kirkpatrick, A.L.Sangiovanni-Vincentelli. “Digital Sensitivity: Predicting Signal Interaction using Functional Analysis”, ICCAD-96, pp.536-541.

[4] F.M.Brown. “Boolean reasoning”, Kluwer Academic Publishers, 1990.

[5] Shepard K.L. “Design methodologies for noise in digital integrated circuits”, Proc., DAC, 1998, pp. 94-99.

[6] A.Rubio, N.Itazaki, X.Xu and K.Kinoshita, “An Approach to the Analysis and Detection of Crosstalk Faults in Digital VLSI Circuits”, IEEE Trans. on CAD, Vol.13, No.3, 1997.

[7] K.L.Shepard. “Design methodologies for noise in digital integrated circuits”, DAC-98, pp.94-99.

[8] Glebov, A., Gavrilov, S., Blaauw, D., Sirichotiyakul, S., Chanhee Oh, Zolotov, V., “False-noise analysis using logic implications”, ICCAD 2001 pp: 515 -521

[9] R.E.Bryant. “Graph-Based Algorithms for Boolean Function Manipulation”, IEEE Trans. on Computers, 1986, v.35, pp.677-691.

[10] J.A.Robinson A. Machine-Oriented Logic Based on the Resolution Principle, Journal of the ACM, 12(1): 23-41, 1965.

[11] A.Glebov, S.Gavrilov, D.Blaauw, V.Zolotov, R.Panda, C.Oh, “False-Noise Analysis using Resolution Method”, ISQED 2002, March 2002.

[12] Kunz, W.; Pradhan, D.K., Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization. IEEE Trans. on CAD Vol. 13 No. 9, Sept. 1994 pp. 1143 -1158