CROSSTALK NOISE AVOIDANCE IN ASYNCHRONOUS CIRCUITS

Alexander Taubin[1] *Alex Kondratyev*[1] *Jordi Cortadella*[2] *Luciano Lavagno*[3]

[1]The University of Aizu, Japan
[2]Univ. Politècnica de Catalunya, Barcelona, Spain
[3]Politecnico di Torino, Italy

ABSTRACT

For deep sub-micron (DSM) technology, noise immunity becomes a matter of crucial importance. The absence of a common clock in asynchronous systems helps to reduce the noise produced by signals switching simultaneously. However for crosstalk noise, an asynchronous approach gives no advantage in comparison to synchronous circuits. This paper concentrates on crosstalk noise and suggests a method for its avoidance in asynchronous circuits. Issues of noise immunity are considered at early design stages starting from the behavioral level. This enables to talk about a design methodology for "low noise" circuits in the same fashion as design for low power or testability. The preliminary experimental results show that noise avoidance through behavioral transformations is highly efficient. Only a small portion of the wires remain as potential victims of noise and are left for for noise avoidance stages at layout level.

1. INTRODUCTION

For DSM technology, noise immunity is becoming a metric of importance comparable to area, timing and power [16, 6]

Immunity against external noise sources (α-particle radiation, substrate noise, etc) can be ensured for DSM by the same means as for VLSI – through an appropriate packaging and electromagnetic isolation. On the contrary, the problems of internal noise immunity (crosstalk, simultaneously switching noise) become much more severe due to the following reasons: increasing capacitive coupling (closer packing, nonuniform geometrical scaling, etc.), lower noise margins, higher speed of voltage changes.

The absence of a common clock in asynchronous systems helps to avoid the simultaneously switching noise [13].

However for crosstalk noise, an asynchronous approach gives no advantage in comparison to synchronous circuits. The straightforward way to reduce noise is to constrain the layout by forbidding the adjacency of "talkative" wires. The drawbacks of this approach is that the information on which wires are subject to crosstalk comes on late design stages. Noise reduction may take several iterations with a layout tool. This may be too laborious and gives no guarantee of convergence. A common trend is to find design flaws (due to noise in particular) as early as possible in a design flow. Here one can talk about design for "low noise" in the same fashion as in design for low power or testability.

An important step into this direction was made in [7, 8]. There it was suggested to extract information about the possible sources of noise at gate level, thus giving a good guidance for layout tools. The target of this paper is to deal with noise issues even earlier, i.e. at behavioral level. The possible advantages are twofold: 1) noise reduction can be achieved by providing local transformations at behavior level (reshuffling of transitions e.g.) 2) it provides a uniform approach for consideration of noise in both sequential and combinational logic, while the methods in [7, 8] were mainly targeted for combinational circuits (analysis of sequential parts was also considered, but not exact).

These techniques can be combined with traditional methods that constrain the future layout, even though the behavioral transformations often make the problem much easier to solve because they partly (and sometimes completely) remove the sources of crosstalk noise.

2. CROSSTALK NOISE AND CIRCUIT FAULTS

There are two types of faults that can be caused by noise.

Definition 2.1 (Noise Generated Faults) *1.* **Transient fault** - *a signal temporally changes its logical value due to noise 2.* **Delay fault** - *a signal delay (e.g. including wire after fork) exceeds the specified worst case*

The source of crosstalk noise lies in the interaction of adjacent wires related to the coupling capacitance between them (it is also called coupling noise).

For a pair of wires **(a, v)** we will call **a** an *aggressor* and **v** a *victim*, if a transition on **a** might produce a noise on **v**. This distinction is a relative one because at the same time both wires can act as aggressors and/or victims.

A noise analysis can be confined to immediate adjacent lines, since they act as shields and protect a "victim" from other "aggressors" [21]. The total crosstalk noise voltage can be computed as a sum of individual noise contributions of each of adjacent aggressors.

Transient and delay faults generated by noise are of different nature and happen in two different scenarios. For a transient fault to occur at a victim line (which should keep a stable value) a short pulse must be generated due to transitions on adjacent aggressor lines. Fortunately a voltage swing of a victim line in response to a transition on a single adjacent line is small enough and can be successfully filtered by the rest of a circuit [6]. However the cumulative effect of two lines switching in the same direction might produce a pulse sufficient to drive the gates connected to a victim line. Therefore, for transition faults we will consider only situations when two or more aggressor wires are switching into the same direction.

A known design solution for crosstalk reduction is to space the tracks farther apart. However, it does not work for DSM. Even with a double minimum spacing, for 70-nm technology the coupling noise caused by two aggressors switching into the same direction can exceed 30% of V_{dd} [4]. Therefore, solving the problem of crosstalk noise in DSM requires a design methodology which cannot be restricted to layout level only. Noise issues must be taken into account at logic and behavioral level as well.

When considering delay faults due to crosstalk noise, one should work on a different scenario. Delay faults are produced by the interaction of wires having simultaneous transitions in opposite directions [7, 8]. The coupling between them may inject charge on the one more weakly driven, thus slowing down its switching.

Faults generated by noise are dangerous only if they can influence the rest of the circuit. In this sense, we can talk about noise generation and noise propagation conditions. To increase noise immunity one can either avoid noise generation or prevent its propagation. These issues are considered in Section 5.

3. BASIC NOTIONS

3.1. Basic definitions about Boolean Functions

An incompletely specified (scalar) Boolean function is a functional mapping $F : B^n \to \{0, 1, -\}$, where $B = \{0, 1\}$ and '$-$' is a *don't care* value. The subsets of domain B^n in which F holds the 0, 1 and don't care values are respectively called the *OFF-set*, *ON-set* and *DC-set*. Let $F(x_1, x_2, \ldots, x_n)$ be a Boolean function of n Boolean variables. A point (i.e. binary vector of n values) in the domain of *ON-set*(F) is called a *minterm*. A variable $x \in X$ is *essential* for function F (or F is dependent on x) if there exist at least two minterms $v1, v2$ different only in the value of x, such that $F(v1) \neq F(v2)$. The set of essential variables for a Boolean function F is called the *true support* of F.

Let $F(X)$ be a Boolean function $F(X)$ with support $X = \{x_1, x_2, \ldots, x_n\}$. The *cofactor* of $F(X)$ with respect to x_i ($\overline{x_i}$) is defined as $F_{x_i} = F(x_1, x_2, \ldots, x_i = 1, \ldots, x_n)$ ($F_{\overline{x_i}} = F(x_1, x_2, \ldots, x_i = 0, \ldots, x_n)$, respectively). The well-known Shannon expansion of a Boolean function $F(X)$ is based on its cofactors: $F(X) = x_i F_{x_i} + \overline{x_i} F_{\overline{x_i}}$.

The existential abstraction of a function $F(X)$ with respect to x_i is defined as $\exists_{x_i} F = F_{x_i} + F_{\bar{x}_i}$. The existential abstraction can be naturally extended to a set of variables. The *Boolean difference*, or *Boolean derivative*, of $F(X)$ with respect to x_i is defined as $\delta F/\delta x_i = F_{x_i} \oplus F_{\overline{x_i}}$.

A characteristic function of a set S of n-dimensional boolean vectors is a boolean function $F : B^n \to B$ such that $s \in S \Leftrightarrow F(s) = 1$.

3.2. Behavioral models and Logic Implementability

In this subsection we assume the reader to be familiar with the Petri net model, a formalism used to specify concurrent systems. We refer to [15] for a general tutorial on Petri nets.

A *State Graph* (SG) is a labeled directed graph whose nodes are called *states*. Each arc of an SG is labeled with an *event*, that is a rising ($a+$) or falling ($a-$) transition of a signal a in the specified circuit. We also use the notation $a*$ if we are not specific about the direction of the signal transition. The set of signals of an SG is called $X = I \cup O$, where I and O denote the set of input and output signals respectively. The behavior of the input signals is determined by the environment whereas the behavior of the output signals must be implemented by the circuit. We write $s \xrightarrow{a}$ ($s \xrightarrow{a} s'$) if there is an arc from state s (to state s') labeled with a.

A labeling function $v : S \to \{0, 1\}^n$ assigns a vector of signal values to each state ($n = |X|$). We will call $v_a(s)$ the value of signal a in state s. The symbol 0^* (1^*) indicates that a rising (falling) transition of the corresponding signal is enabled in that state. An SG is *consistent* if:

$$
\begin{aligned}
s \xrightarrow{a+} s' &\implies v_a(s) = 0 \ \wedge \ v_a(s') = 1 \\
s \xrightarrow{a-} s' &\implies v_a(s) = 1 \ \wedge \ v_a(s') = 0 \\
s \xrightarrow{b*} s' \ \wedge \ a \neq b &\implies v_a(s) = v_a(s')
\end{aligned}
$$

A *Signal Transition Graph* (STG) is a Petri net in which transitions are labeled with the same type of events we have defined for SGs, i.e. rising and falling signal transitions [3, 11].

An STG has an associated SG in which each reachable marking corresponds to a state and each transition between a pair of markings to an arc labeled with the same event of the transition.

Although STGs with bounded reachability space and SGs have the same descriptive power, STGs can usually express the same behavior more succinctly.

Figure 1.a depicts an STG with three signals. For simplicity, places with only one input and output transitions are omitted. Figure 1.b shows the corresponding SG with states labeled with the binary vectors of the signal values. The SG is consistent.

In addition to consistency, *speed independence* and *Complete State Coding* (CSC) are two properties required for an SG to be implementable as a hazard-free asynchronous circuit [9].

4. DIGITAL SENSITIVITY CONCEPT AND NOISE AVOIDANCE

The digital sensitivity approach in [7, 8] suggests a constraint-driven design methodology in which the information extracted from logical or timing correlations between signals is provided to a layout synthesis tool (a constraint-based channel router). In the simplest case, this information should indicate which sets of wires are unsafe to be routed adjacently because of the possibility for crosstalk noise. In the most complex case, it gives a set of layout constraints (usually conservative) that guarantee the circuit to be tolerant to noise when satisfied.

From the discussion of Section 2. on the sources of noise faults, it follows they are always produced by specific cases of several signals switching simultaneously. Therefore conditions for noise generation can be checked by concurrency relations between signal transitions. Information on concurrency is explicitly represented in the behavioral model. In particular two transitions $a*$ and $b*$ are concurrent ($a* || b*$) if they are enabled in the same state of the SG and firing one of them does not disable the other. Formally, for $a* = a+$ and $b* = b+$ e.g., $a+$ concurrent to $b+$ means that there exists a state s in the SG such that: $s \xrightarrow{a+} s_1 \ \wedge \ s_1 \xrightarrow{b+} \ \wedge \ s \xrightarrow{b+} s_2 \ \wedge \ s_2 \xrightarrow{a+}$.

The following SG-based algorithm identifies sets of wires that might potentially be sources of noise faults. This algorithm formalizes the conditions of occurrence of noise faults which were presented in Section 2.

```
Input:  Initial SG A
Output: Set Noise of potentially noisy wires
    foreach s ∈ A do
        /* Check delay faults generated by noise */
        foreach pair a,b s.t.
        v_a(s) = 1* ∧ v_b(s) = 0* ∧ a-||b+ do
            Del_faults = Del_faults ∪ {a, b};
        /* Check transient faults generated by noise */
        foreach triple a,b,c s.t.
        v_a(s) = 1* ∧ v_b(s) = 1* ∧ v_c(s) = 1 ∧ a-||b- do
            Trans_faults = Trans_faults ∪ {a, c, b};
        foreach triple a,b,c s.t.
        v_a(s) = 0* ∧ v_b(s) = 0* ∧ v_c(s) = 0 ∧ a+||b+ do
            Trans_faults = Trans_faults ∪ {a, c, b}
    enddo
    Noise = Trans_faults ∪ Del_faults
```

Figure 2. Analysis for noise fault

Example 4.1 *Let us apply the algorithm in Figure 2 to the xyz example in Figure 1. In the corresponding SG there are only two states in which signals can fire concurrently: 11*0* and 1*1*1. The former may derive a possible delay fault by the pair of wires $\{y, z\}$, while the latter may be the source of a transient fault for the triple $\{x, z, y\}$, where z is a victim wire and x, y are the aggressors (for a triple the order of signals is important – victim wire is always placed between aggressors). Hence in this example $Noise = \{\{y, z\}, \{x, z, y\}\}$.*

For those signals that successfully pass the check about noise faults (i.e. for those that are not included in the *Noise* set) we will talk about *logic separation*. Hence, so far the notion of logic separation of signals is based completely on the analysis of the ordering/concurrency relations between transitions in the behavioral (STG) specification. However, two transitions that are concurrent in the specification do not necessarily happen ("fire") concurrently in the circuit. This is because delays of circuit gates and wires may actually cause two signals that are specified as concurrent (e.g., pairs of gates triggered by the same transition) to actually occur always in a specific order. For this reason, the concurrency relation in the specification is a conservative approximation of reality.

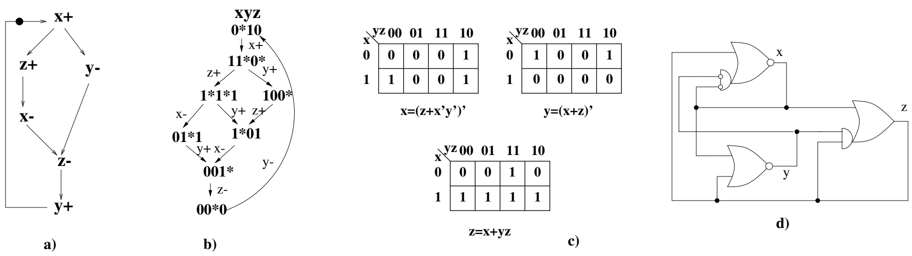

Figure 1. (a) STG, (b) SG, (c) next-state functions, (d) complex-gate implementation.

Refinement of concurrency is thus possible, either by considering the implementation (when delay estimation becomes possible) or by making some "reasonable" timing assumptions, which must then be satisfied (e.g., by transistor sizing or constrained routing) by the implementation. If signals are not logically separated but, according to these timing informations, we can conclude that they cannot switch simultaneously, we will talk about *temporal separation*.

Temporal separation can be checked in a way similar to the algorithm of Figure 2. In [20, 10] it was shown that timing constraints can be captured at the STG level, by introducing timing arcs that reduce the potential concurrency of signals. From such an STG with timing arcs one can generate the SG and apply the same algorithm to it. The resulting $Noise$ set will contain tuples of signals that do not have logical nor temporal separation.

A further refinement of these sets of potentially noisy signals could be obtained by considering the *noise propagation conditions*. If some signals are not separated (logically or temporally) and could generate a crosstalk noise, we would still have no problems if the generated noise faults could not propagate through the circuit. Suppose, for example, that a transient fault on wire a happens at a time when for all fanout gates of a some input different from a has a controlling value. Then the behavior of the fanout of a does not actually depend on the value a, and any transient fault affecting a could not propagate within the circuit (the formal conditions of noise propagation are considered in Section 5.).

If for a set of signals the conditions for noise generation or propagation are not satisfied, then the set is said to be *isolated from crosstalk noise*. The level of *digital isolation* from crosstalk noise is the percentage of signal pairs for which the circuit is isolated from crosstalk noise (*isolated pairs*), considering each signal as a victim candidate.

According to [8], about 50% of digital isolation is needed to have a reasonable impact on the reduction of circuit area after layout, due to a smaller set of layout constraints.

The main improvement introduced by this work with respect to the contribution of [7, 8] is the consideration of behavioral *sequential* models in the digital sensitivity approach. The STG- and SG-based methods that we suggest can be useful both for efficient derivation of isolated pairs and for increasing the digital isolation through specification transformations.

In Section 8. it will be shown that the combination of behavioral transformations together with reasonable timing assumptions allows us to reach about 80% of digital isolation in average.

The methodology for low-noise asynchronous circuit design includes the following steps:

1. Analysis of noise-critical sets of signals.

   This reduces to checking logical and temporal separation on a behavioral specification, and providing the layout tools with constraints on adjacency of critical wires.

2. Improving logical and temporal separations of signals.

   This might be achieved by *optimizations of behavioral specifications* aimed at noise avoidance. These optimizations include: (a) signal reshuffling and concurrency reduction [5] (logical separation) and (b) timing assumptions [10] (temporal separation).

3. Logic optimization for low noise.

   Logic functions for gates might be chosen according to a cost function evaluating noise. This requires different approaches to minimization (e.g., introducing redundancy in the circuit in order to avoid noise propagation). Similar ideas were exploited in an alternative wiring scheme from [12].

4. Clustering for reduction of crosstalk noise.

   Interconnections could be divided into local and global levels. The global level (top layers of interconnect structure [2]) should include all (or most of) the long wires that could be dangerous from the point of view of crosstalk just because of the long cross couplings. The local level (lower layers of interconnect) should be used only for short distance interconnects, but could still suffer from crosstalk noise because of high density (closer packing and nonuniform scaling). We could potentially exploit this fact and divide large circuits into relatively small "islands", connected with each other by global and vertical interconnects. Crosstalk noise analysis and synthesis could be done separately for each "island", that could be small enough to apply our behavior specification techniques.

The techniques presented in the paper mainly concern items 1, 2 and 3 from the list above. Item 4 gives a more systematic approach for the avoidance of noise, but it is still an open area for research.

5. DIGITAL SENSITIZATION ANALYSIS

The analysis of logical or temporal separation[1] requires one to consider each signal as a potential victim, and identify sets of states in which the signals might be a subject to noise faults.

Definition 5.1 (Fault Sets) *1. The* **transient fault set** *with respect to signal b (denoted by $S_{tr}(b)$) is the set of SG states such that $\forall s \in S_{tr}(b)$ there are two enabled signals a and c, which have the same value as b.*

2. The **delay fault set** *with respect to signal b (denoted by $S_{del}(b)$) is the set of SG states such that $\forall s \in S_{del}(b)$, b is enabled and there exists another signal a enabled in s, with the opposite logical value to b.*

3. The **noise fault set** *with respect to signal b (denoted by $S_n(b)$) is the union of the transient and delay fault sets.*

Transient fault propagation. The formal conditions of transient fault propagation can be formulated in terms of *sensitization* [14].

Sensitization of a gate g with respect to signal b captures the conditions under which the value at a gate output depends on the value of b. Formally, when g implements Boolean function F, its sensitization with respect to b is: $Sens_F(b) = \frac{dF}{db}$.

The inverse of gate sensitization (observability don't care, ODC) gives conditions under which the value on a wire cannot affect a gate output. In particular, if a transient fault on wire b occurs in a circuit state in which gate g is not sensitized to b, then the fault cannot propagate through the gate. The idea of

[1] As mentioned in Section 4., temporal separation can be reduced to logic separation by considering the SG obtained from an STG with added timing constraints.

using gate sensitization for checking noise fault propagation in a circuit was first suggested in [8].

Definition 5.2 (Digital sensitivity) *Digital sensitivity of a gate g to noise faults in a wire b ($DS_g(b)$) is the conjunction of the characteristic function of the Noise fault set $S_n(b)$ and of the function: $DS(b) = S_n(b) \cap Sens_g(b)^2$.*

Digital sensitivity of a circuit to noise faults on a wire b ($DS(b)$) is obtained by the union of the digital sensitivity conditions of all the gates in the fanout of b.

If $DS(b)$ is empty, then the circuit is insensitive to faults occurring on a victim wire b. Hence all the aggressor-victim tuples that could generate these faults are actually *isolated from crosstalk noise*.

Example 5.1 *For the example in Figure1.b, the sets of wires that may be the sources of crosstalk noise are: $Noise = \{(y,z),(x,z,y)\}$. The only state in which a transient fault may occur is $S_{tr}(z)$= {1*1*1}. Let us check, by only using the sensitivity conditions for wire z, whether the transient fault can propagate through the circuit.*

The functions of the individual gates in this example are:

$$x = \overline{z + \overline{xy}} = \overline{z}x + \overline{z}y \qquad y = \overline{x+z} = \overline{xz} \qquad z = x + yz$$

From them we can calculate the corresponding sensitivity functions:

$$\begin{aligned}
Sens_x(z) &= x_z \oplus x_{\overline{z}} = 0 \oplus (x+y) = x+y \\
Sens_y(z) &= y_z \oplus y_{\overline{z}} = 0 \oplus \overline{x} = \overline{x} \\
Sens_z(z) &= z_z \oplus z_{\overline{z}} = (x+y) \oplus x = \overline{x}y \\
Sens(z) &= Sens_x(z) + Sens_y(z) + Sens_z(z) = 1
\end{aligned}$$

The circuit is sensitive to a fault on wire z in every SG state. Hence the considered transient fault may indeed propagate through the circuit.

The results of propagation analysis hence in this case *cannot reduce the Noise set. We have to assume that both tuples (y,z) and (x,z,y) might be a source of noise faults.*

6. NOISE ANALYSIS USING STG AND SG

6.1. Transient faults

When discussing digital sensitivity (Section 5.), we made several simplifications that need to be justified.

Simplification 1. The suggested method treats transient faults *one at a time*, assuming that faults are not influencing each other. We neglect the effects of fault interference, because the probability of two glitches arriving at the same time to the inputs of a gate seems to be rather small. This is similar to the assumptions that are traditionally made in testing, when considering single fault models versus multiple fault models [1].

Simplification 2. Sensitivity analysis in general assumes that sensitivity conditions are calculated when an input set is applied to the circuit, and then the circuit has enough time to stabilize. This is like the standard *fundamental mode* assumption. However, circuits synthesized from an STG operate in I/O mode, which means that inputs might change while part of the circuit is still reacting to a previous input set.

Therefore, we need to discuss the assumptions under which this "static" view is applicable to a circuit operating in I/O mode.

Let us start from an example, and consider the STG in Figure 3.a. One can derive the implementation of its signals k and e shown in Figure 3.c.

In the corresponding SG (Figure 3.b) the only state which might be the cause of a transient fault is 1*1*100. This fault produces a pulse on victim wire c and the propagation of this pulse could be checked by a sensitivity analysis of the NOR gate k with respect to c: $Sens_k(c) = \overline{a} \;\Rightarrow\; DS_k(c) = S_{tr}(c) \cdot Sens_k(c) = abc\overline{k}\overline{e} \cdot \overline{a} = 0$

[2] If the meaning is clear from the context, we will liberally identify the characteristic function of a set and the set itself.

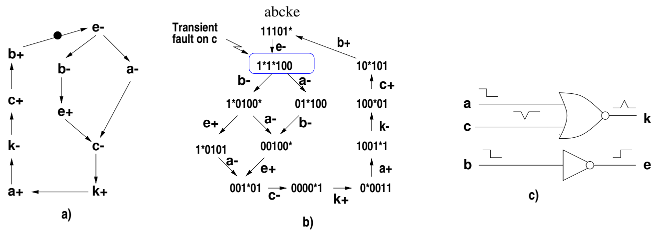

Figure 3. Transient fault and its propagation

From $DS_k(c) = 0$ one can conclude that the transient fault on wire c is not propagating through gate k because it is blocked by the controlling value 1 on wire a. However, in its own turn signal a is switching in state 1*1*100. Therefore, the propagation of a transient fault may depend on the relative propagation speed of the fault and some other transition (in this case $a-$ propagating to the inputs of gate k). If we denote by $wire\_delay(a,k)$ and $wire\_delay(c,k)$ the delays of the wires between gates a and k and c and k respectively, then the blocking of a transient fault on wire c requires $wire\_delay(a,k) > wire\_delay(c,k)^3$. This condition may be difficult to check, because of a number of reasons. It requires precise control over/analysis of wire delays, the inequalities could be inconsistent for different transient faults, etc. Therefore, our current sensitivity analysis might be *too optimistic* by not considering propagation delays.

The example in Figure 3 shows that signals which are enabled in a state that causes a transient fault are poor candidates for blocking the fault propagation. This leads to the notion of *static sensitivity* with respect to a state (the word "static" indicates that sensitivity conditions are stable at least for the considered state and its immediate successors).

Definition 6.1 (Static sensitivity) *Let an SG state s be the source of a noise fault in wire b and let $En(s)$ denote the set of signals enabled in s. The static sensitivity of gate g to the noise fault in state s ($SS_g(b,s)$) is given by the product of the characteristic function for s and the existential abstraction of the sensitivity function with respect to signals in $En(s)$: $SS_g(b,s) = s \cdot \exists_{a \in En(s)} Sens_g(b)$.*

Static sensitivity assumes that checking for transient fault propagation is performed for each state separately. When state s is the source of a transient fault in a victim wire b due to aggressors a and c and $SS_g(b,s) \neq \emptyset$ the corresponding noise triple (a,b,c) is considered to be dangerous.

Example 6.1 *Consider the SG in Figure 3.b. In state $1*1*100$ the set of enabled signals is $\{a,b\}$. $Sens_k(c) = \overline{a} \Rightarrow \exists_{x \in \{a,b\}} Sens_k(c) = 1$. Moreover, $SS_k(c, 1*1*100) = abc\overline{k}\overline{e} \cdot 1 \neq \emptyset$, the transient fault on c may propagate through gate k, and hence noise triple $\{a,c,b\}$ should be considered as dangerous.*

The idea behind the application of static sensitivity analysis instead of digital sensitivity (Definition 5.2) is to extend its applicability to *asynchronous sequential circuits*, by avoiding unrealistic (or difficult to implementation) timing assumptions about wire delays at the fanins of a gate. Let us consider Figure 4 and derive the timing assumptions behind static sensitivity. Suppose that in state s a transient fault is generated at input c of gate Y, and that the fault does not propagate through Y according to the static sensitivity conditions.

The latter means that some the inputs of gate Y ($fanin(Y)$) have logical values that block propagation of a pulse on c to the output of Y. Moreover, due to Definition 6.1, these blocking signals are stable in state s. The blocking signals may, however, change their values when the gates in $fanin(Y)$ will make transitions (e.g., gates $X, Z, \ldots$). These transitions in their

[3] In fact, instead of delay $wire\_delay(c,k)$ we should consider the delay from the source of noise in wire c to the input of gate k. However, since that point is not known until after layout, we conservatively use $wire\_delay(c,k)$.

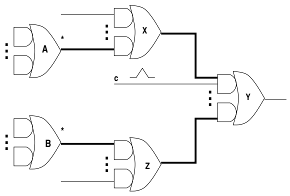

Figure 4. Timing assumptions for static sensitization

own turn should be caused by changes of signals in $fanin(X)$, $fanin(Z), \ldots$ (e.g., A, B).

Let us estimate the earliest time when blocking signals can change their values after the circuit starts from state s. In the worst case scenario (from the timing point of view) the signals causing transitions in $X, Z, \ldots$ are enabled in s (otherwise, the blocking signals at $fanin(Y)$ will change even later). Then their propagation paths are shown in bold in Figure 4. If we denote by $min\_wire(fanin(V))$ the minimum delay among the wires in $fanin(V)$ and by $min\_gate(fanin(V))$ the minimum delay of gates in $fanin(V)$, then the minimum delay of these paths would be: $min_{V \in fanin(Y)}(min\_wire(fanin(V))) + min\_gate(fanin(Y)) + min\_wire(fanin(Y))$. From here we can derive the worst case timing assumption that limits the applicability of static sensitivity: $min_{V \in fanin(Y)}(min\_wire(fanin(V))) + min\_gate(fanin(Y)) + min\_wire(fanin(Y)) > wire\_delay(c, Y) \quad (T1)$
This means that a transient fault in wire c will not propagate through gate Y if the wire delay of (c, Y) is less than *the time needed to change the value of the static sensitivity function.* Notice that:

1. Timing assumption (T1) can always be satisfied in an implementation by delay padding of the gates in $fanin(Y)$. Delay padding of a gate cannot affect the validity of other timing assumptions (due to consideration of states different from s), because the right hand side (T1) contains only a wire delay, not a gate delay. This gives simple proof of convergence of the delay padding procedure. The requirement (T1) is very similar to the conditions of hazard free implementation from [11], where various delay padding algorithms are considered in detail.
2. However, (T1) is very difficult to satisfy by gate insertion or transistor sizing if wire delays dominate over gate delays. In this case, only relatively expensive constrained routing, with the goal of minimizing wire delay skew [18], can be used. Hence it becomes desirable to refine (T1) and make it less conservative, if possible.

This refinement of timing assumptions can be performed using the following procedure *Refine_assumptions*:

1. Start from state s that is the cause of a transient fault on wire c and such that $SS_g(c, s) = 0$.
2. Traverse the SG up to the states $s' \in S1$ in which $SS_g(c, s') \neq 0$.
3. For each $s' \in S1$ estimate the path delay for reaching s' from s ($del\_path(s, s')$) as the sum of wire and gate delays (for non-concurrent transitions).
4. Choose the minimal path delay: $del\_path\_min = min_{s' \in S1}(del\_path(s, s'))$
5. For each gate y in the fanout of c add a timing constraint $del\_path\_min < wire\_delay(c, y)$

6.2. Delay faults

In the previous Section we discussed the conditions under which transient faults are filtered by the circuit. They are based on static sensitivity analysis together with timing assumptions which justify the analysis. This technique can also be extended for the analysis of delay faults. We briefly sketch how to apply it.

Delay faults produce changes in the temporal behavior of some events. These changes may be dangerous if some timing constraints are assumed for the correct behavior of the circuit. Unfortunately, any circuit not in the class of delay-insensitive circuits is vulnerable to delay faults.

Given that speed-independent circuits are correct under any gate delay, delay faults may only affect wires with isochronic forks [19]. If a wire v has a symmetric isochronic fork, a delay fault with $v*$ as a victim is always dangerous, no matter which branch of the fork it affects. In case of asymmetric isochronic forks, delay faults are only dangerous if they affect the fast branch.

A detailed analysis of different cases for delay faults is provided in [17]. Here we will give only a summary of it.

1. There are exceptional cases when delay faults can be neglected
(E1) Delay fault happens at a wire which has no fork
(E2) If a transition $v*$ in STG triggers only transitions of input signals then under a hypothesis of "slow environment" any delay fault on the wires originating at gate v and feedbacking to a circuit are safe. (The latter corresponds to an asymmetric isochronic fork by wire v, where its fast branch is a branch which goes to environment)
2. For checking the propagation of delay faults through branches of isochronic forks one can apply an analysis based on sensitization. (This analysis is similar to the case of transient faults with the only difference that in it we should consider a digital sensitization instead of the more complicated static sensitization.)
3. An analysis of delay faults propagation shows that under a conservative timing assumptions (similar to (T1)) *no delay fault* propagates through a circuit. Better (than (T1)) timing assumptions might be obtained via Procedure *Refine_assumptions*.

Based on analysis of noise faults generation and propagation we can suggest two extreme approaches for checking a noise isolation.

- *Optimistic approach.* We assume that any timing constraint which is derived from a sensitization analysis for noise faults might be satisfied in an implementation. Hence transient faults which are not sensitized and **all** delay faults are neglected. The remaining faults are reported to layout tool as dangerous ones.
- *Conservative approach.* No timing assumptions are used for separating faults on observable and not. Under this conservative view all transient faults and all delay faults (excluding exceptional cases (E1) and (E2)) are assumed to be dangerous.

In Section 8. we show the results of noise isolation only for optimistic and conservative approaches. However in practice it might be reasonable to use an intermediate solution which lies between these two extreme cases. For example we could assume the existence of a small set of timing constraints (either given a priori or produced by timing analysis) and only part of faults is filtered due to these constraints.

7. IMPLEMENTATION ISSUES

The methods proposed in this paper have been implemented in the tool **petrify**. For preliminary experiments, the methods have been applied only to the synthesis of speed-independent circuits.

Two different aspects have been considered in the implementation: analysis and synthesis.

Analysis has been reduced to the calculation of those wires subject to noise faults, according to the algorithm presented in Figure 2. The analysis has been enhanced by [4]

- calculating the static sensitization of wires to report only those transient faults that may be propagated (see definition 6.1) and

[4] The technical report [17] contains a complete algorithm for the noise analysis

- by doing a conservative estimation of the wire forks of the circuit, according to the case study presented in section 6.2.. The estimation is conservative in the sense that no timing assumptions are done on the delay of the gates.

Synthesis. A method for synthesis aiming at the reduction of noise faults has also been implemented. The method is based on behavioral transformations of the specification to reduce the concurrency events that may act as agressors. The algorithm consists of the following steps:

1. Derive a complex-gate implementation of the circuit
2. Analyze the circuit, as explained above, and derive the set of noise faults.
3. Create a list of "noisy" pairs of events with one pair $(a*, b*)$ for each noise fault produced by $a* \parallel b*$.
4. Transform the specification aiming at reducing the concurrency of the noisy pairs of events.
5. Synthesize the circuit.

The algorithm has been integrated in the phase of concurrency reduction of `petrify` [5]. The cost function has been biased towards reducing the concurrency of those pairs of events that are "noisy".

8. EXPERIMENTAL RESULTS

We made experiments with a known set of asynchronous benchmarks. The results are presented in [17].

The experiment was organized as follows.

- We implemented all the circuits by atomic complex gates, since the library is not crucial for the problem of noise avoidance.
- We used the crosstalk analysis algorithm to discover dangerous (noisy) concurrent states, isolated pairs and noisy wire configurations for both conservative and optimistic assumptions.
- We performed concurrency reduction for all dangerous concurrent states and used the crosstalk analysis algorithm (for almost all examples one reduction was enough).

From these *preliminary* experiments we can conclude that crosstalk noise avoidance using behavioral specifications is a very promising technique for asynchronous control circuits, because:

- it can effectively extract all isolated and noisy wire configurations from the behavioral specification, and thus supply a set of constraints that can completely avoid crosstalk noise to the layout tools.
- it simplifies the task of constrained layout generation by using noise avoidance techniques at behavioral and logic level.

These experiments showed that about 80% of noise isolation can be reached in average by behavioral transformations. Previous work [5] has confirmed that restricted behavioral transformations (concurrency reduction e.g.) allow the designer to save up to 20% of area under a small degradation of performance (below 5%). This gives a ground that, in the suggested methodology targeted for "low noise", the quality of implementation is not significaltly degraded. In fact, the area of the circuits after noise avoidance has decreased in all our benchmarks.

9. ACKNOWLEDGEMENTS

This work has been partially funded by CICYT TIC 98-0410.

REFERENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press, 1990.

[2] M. Bohr. Silicon trends and limits for advanced microprocessors. *Communications of the ACM*, (3):80–87, 1998.

[3] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications.* PhD thesis, MIT, June 1987.

[4] J. Cong. Deep submicron layout and coupling to logic synthesis (invited talk). In *International Workshop on Logic and Architecture Synthesis (IWLAS'97)*, 1997.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Automatic handshake expansion and reshuffling using concurrency reduction. In *Workshop on Hardware Design and Petri Nets*, pages 86–110, June 1998.

[6] L. Gal. On-chip cross talk – the new signal integrity challenge. In *IEEE 1995 Custom Integrated Circuits Conference*, pages 251–254, 1995.

[7] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli. Techniques for crosstalk avoidance in the physical design of high-performance digital systems. In *Proceedings of the International Conference on Computer-Aided Design*, 1994.

[8] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli. Digital sensitivity: Predicting signal interaction using functional analysis. In *Proceedings of the International Conference on Computer-Aided Design*, 1996.

[9] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky. *Concurrent Hardware. The Theory and Practice of Self-Timed Design.* John Wiley and Sons Ltd., 1994.

[10] Alex Kondratyev, Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, Alexander Taubin, and Alex Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *Proceedings of the International Conference on Computer-Aided Design*, November 1998. Accepted for publication.

[11] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits.* Kluwer Academic Publishers, 1993.

[12] T.-C. Lee and J. Cong. The new line in ic design. *IEEE Spectrum*, (3):52–58, 1997.

[13] W. A. Lien, P. Day, C. Faransworth, D. L. Jackson, J. Liu, and N.C. Paver. Noise in a self-timed and synchronous implementation of a dsp. Unpublished, White Paper, Cogency Technology Inc., http://www.cogency.com/noise.html, 1997.

[14] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., 1994.

[15] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, April 1989.

[16] K.L. Shepard and V. Narayanan. Noise in deep submicron digital design. In *Proceedings of the International Conference on Computer-Aided Design*, 1996.

[17] A. Taubin, A. Kondratyev, J. Cortadella, and L. Lavagno. Behavioral transformations to increase noise immunity in asynchronous specifications. Technical Report 98-2-005, University of Aizu, Japan, October 1998.

[18] R.-S. Tsay. An exact zero-skew clock routing algorithm. *IEEE Transactions on Computer-Aided Design*, 12:242–249, 1993.

[19] K. van Berkel. Beware the isochronic fork. *Integration, the VLSI journal*, 13(2):103–128, June 1992.

[20] Peter Vanbekbergen, Gert Goossens, and Bill Lin. Modeling and synthesis of timed asynchronous circuits. In *Proceedings of the European Design Automation Conference (EURO-DAC)*, pages 460–465, September 1994.

[21] A. Vittal and M. Marek-Sadowska. Crosstalk reduction for VLSI. *IEEE Transactions on Computer-Aided Design*, (3):290–298, 1997.

CROSSTALK NOISE AVOIDANCE IN ASYNCHRONOUS CIRCUITS

Alexander Taubin[1] , *Alex Kondratyev*[1] , *Jordi Cortadella*[2] *and Luciano Lavagno*[3]

[1]The University of Aizu, Japan

[2]Univ. Politècnica de Catalunya, Barcelona, Spain

[3]Politecnico di Torino, Italy

For deep sub-micron (DSM) technology, noise immunity becomes a matter of crucial importance. The absence of a common clock in asynchronous systems helps to reduce the noise produced by signals switching simultaneously. However for crosstalk noise, an asynchronous approach gives no advantage in comparison to synchronous circuits. This paper concentrates on crosstalk noise and suggests a method for its avoidance in asynchronous circuits. Issues of noise immunity are considered at early design stages starting from the behavioral level. This enables to talk about a design methodology for “low noise” circuits in the same fashion as design for low power or testability. The preliminary experimental results show that noise avoidance through behavioral transformations is highly efficient. Only a small portion of the wires remain as potential victims of noise and are left for for noise avoidance stages at layout level.