Externally Hazard-Free Implementations of Asynchronous Circuits*

Milton Sawasaki Chantal Ykman-Couvreur Bill Lin

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract — We present a new sum-of-product based asynchronous architecture, called the N-SHOT architecture, that operates correctly under internal hazardous responses and guarantees hazard-freeness at the observable non-input signals. We formally prove that within this architecture a very wide class of semi-modular state graphs with input choices (either distributive or non-distributive) that satisfy the complete state coding property always admit a correct implementation. As with synchronous circuits, we permit internal hazards in the combinational logic core, which means we can make use of conventional combinational logic minimization methods to produce the sum-of-product implementation. This represents a significant departure from most existing methods that require the combinational logic to be hazard-free and are mainly valid for distributive behaviors.

I Introduction

The design of asynchronous circuits is a difficult task since circuit malfunction can occur during execution if the circuit is not hazard-free. In this paper, we consider the problem of producing gate-level asynchronous circuits for both *distributive* and *non-distributive* behaviors from state graph specifications, that operate correctly under internal hazardous responses and guarantee hazard-freeness at the externally observable non-input signals (i.e. output and state signals).

We present a new sum-of-product based architecture, called the N-SHOT architecture, for tackling this problem. We formally prove that within this architecture a very wide class of semi-modular state graphs with input choices that satisfy the complete state coding property always admit a hazard-free implementation. Our approach can uniformly handle both distributive and non-distributive specifications. Handling non-distributive specifications is important because they arise in practical industrial designs (cf. Section V for examples). The architecture that we propose for producing a hazard-free implementation for any non-input signal consists of sum-of-product (SOP) logic implementations for its set and reset functions, an acknowledgement scheme with a local delay compensation to ensure correct restoring of transitions of this non-input signal, and a new flip-flop element for robust electrical operation. *In our approach, the logic implementation for the set and reset functions need not be hazard-free, which means any existing two-level logic minimization method can be used to synthesize the SOP logic without restrictions. We believe this represents a significant contrast to virtually all existing gate-level synthesis methods that require the combinational logic to be hazard-free.* Our framework is formulated at the state graph level, which means it is widely applicable to a broad class of high-level formalisms [2, 17, 18, 6, 7, 20] that can be semantically defined at the state graph level (e.g. the widely used Signal Transition Graph model [2]). Our method only requires the state graph to satisfy the complete state coding property, which is the minimal requirement necessary to derive unambiguously consistent logic and to satisfy the trigger requirement, which is practically always satisfied.

Our paper is organized as follows. Section II presents an overview of related work. Section III defines the state graph model and the basic concepts that are needed to characterize our method. Section IV presents the new architecture, called the N-SHOT architecture, and its properties. We characterize the class of state graphs that can be implemented with this new architecture and present the synthesis procedure. A key point to our approach is that the logic minimization step is reduced to a conventional logic minimization problem (as in the synchronous case). Our method has been automated in the ASSASSIN compiler [21]. We have carefully simulated a number of synthesized designs at the gate- and transistor- level (using SPICE) to ensure the circuits produced by our method are both functionally and electrically correct. In Section V, we present experimental results for both distributive and non-distributive examples and compare our results on the distributive ones with several available methods.

II Related Work

Several methods exist for producing hazard-free circuits. Some approaches like those described in [20, 8] rely on a *fundamental mode* of operation to simplify the hazard-free implementation problem, but these methods cannot handle the generality of models like Signal Transition Graphs that permit concurrent input-output changes.

One existing approach is based on bounded gate / wire delay assumptions and relies on delay information to eliminate hazards [5]. The current state of this method is restricted to distributive behaviors. Other existing approaches aim at producing speed-independent circuits, which are circuits that will work correctly regardless of the individual gate and wire delays, but the skew of all wire delays at a multiple fanout point is assumed to be negligible. Several methods [2, 7, 17] generate speed-independent circuits on the assumption that each non-input signal can be implemented with one single complex gate; this complex gate is assumed to contain no internal hazards.

One practically useful approach is to use an architecture that consists of sum-of-product (SOP) logic implementations for the up- and the down-excitation functions and asynchronous latches (C-element or RS) for restoring the primary non-input signals of the specification. Beerel and Meng [1] have developed an approach using this architecture, based on a basic method originally developed by Varshakvsky et al. [17], to produce speed-independent circuits. Their approach is based on the use of heuristics and rules to eliminate hazards by restricting logic transformations or inserting additional circuitry. However, their method cannot guarantee a correct solution in all cases and is restricted to distributive specifications. Recently, another approach was proposed in [4], also based on the same architecture, that formalizes the requirement that a state graph has to satisfy in order to produce a hazard-free implementation. This requirement, called the monotonous cover property, requires additional state signals to ensure that each up- and down- transition can be separately implemented by a single *monotonous* AND-gate. Ensuring this property can be a hard task. This method is also restricted to distributive specifications.

*This research was sponsored in part by the European Commission under the ESPRIT (6143) project "EXACT".

32nd ACM/IEEE Design Automation Conference ®

In [9], another method is presented that makes use of synchronizers and a locally generated clock to solve the hazard problem. In this method, all external inputs and feedback state signals must be bounded by a synchronizer called a Q-flop. This can be very expensive in area since the number inputs is typically much more than the number of feedback state signals, thus requiring many more memory elements than our solution or other existing approaches based on the use of C-elements or RS latches [1, 17, 4]. Moreover, the internal clocking scheme requires a tree of N C-elements to implement a N-way rendezvous, where N is again the number of external inputs *plus* the number of feedback state signals. The locally generated clock is produced by inserting a delay line that is at least as long as the longest path through the combinational circuit, which means the circuit has to operate in steps that are at least as slow as the worst-case delay through the combinational logic. The performance is further hampered by the delay through the N-way rendezvous scheme, which can be significant if N is large. Also, the implementation area of the delay lines is significant. Thus, comparing to our approach and other existing approaches like [1, 17, 4, 5], this approach can be significantly more expensive in terms of both area and performance.

III State Graph Model

A State Graphs

A state graph (SG) is a finite automaton given by $G = \langle X, S, T, \delta, s_o \rangle$, where the components are defined as follows.

$X = X_I \cup X_O$ is the set of *signals*, X_I is the set of *input* signals, X_O is the set of *non-input* signals and $X_I \cap X_O = \emptyset$.

$T = T_I \cup T_O$ is the set of *signal transitions*, T_I is the set of *input* signal transitions and T_O is the set of *non-input* signal transitions. (Non-input signals refer to both *external output signals* as well as *internal state signals*.) Each transition can be represented as $+x_j$ or $-x_j$ for the j-th $0 \rightarrow 1$ or $1 \rightarrow 0$ transition of signal x. $*x_j$ denotes either a "$+x_j$" transition or a "$-x_j$" transition. Sometimes index j can be omitted.

S is the set of states and $s_0 \in S$ is the *initial state*. $\delta : S \times T \mapsto S$ is a partial function representing the *transition function* such that if $\delta(s,t) = s'$ is defined, then t is said to be *enabled* in state s and the *firing* of t takes the system from s to s'. This is denoted as $s \xrightarrow{t} s'$ or simply as $s[t\rangle$. The *firing* of a *sequence* of transitions $\sigma = t_1 t_2 \ldots t_m$ is denoted as $s \xrightarrow{\sigma} s'$ or simply as $s[\sigma\rangle$.

Each state $s \in S$ is labelled with a *binary vector* $\langle s(1), s(2), \ldots, s(n) \rangle$ according to the signals $X = \{x1, x2, \ldots, xn\}$ of the system. The labeling is given by a *state assignment function* $\lambda_X : S \times X \mapsto \{0,1\}$. For a given state $s \in S$, $s(i)$ denotes the value of signal xi in state s. For $s, s' \in S$ and $t \in T$ such that $s \xrightarrow{t} s'$, the state assignment function is defined as follows: (1) if $s(i) = 0 \wedge t = +xi$ then $s'(i) = 1$; (2) else if $s(i) = 1 \wedge t = -xi$ then $s'(i) = 0$; (3) otherwise, $s'(i) = s(i)$. If the states of the SG can be encoded according to the above rules, then the SG is said to have a *consistent state assignment*. In a state $s \in S$, if a transition $t = *xi$ is enabled, the corresponding signal xi is said to be *excited* (which can be denoted as $s(i)*$ in the binary code of s) and if it is not excited, it is said to be *stable*.

A state in the SG captures the state of all signals in a circuit, while a transition between states is a transition of exactly one signal. There may be many signals enabled in a state, but exactly one signal transition is fired at a time. This corresponds to the *interleaved concurrency* model.

An example of an SG is shown on Figure 1. Here signals a and b are the input signals and signal c is the output signal.

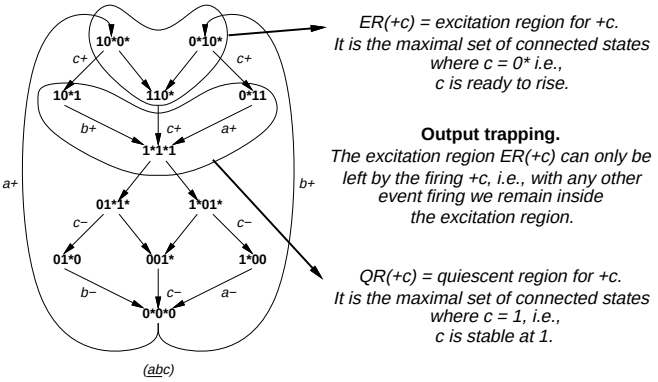

Figure 1: SG example

B Properties and Objects of SGs

We now formally define some basic properties and objects of SGs which are needed in the characterization of our method. We first recall the *Complete State Coding* (CSC) property [2] which is the necessary and sufficient condition to the existence of a race-free implementation for any non-input signal.

Definition 1 (CSC) *An SG satisfies CSC if and only if* $\forall s, s' \in S$ *either they have different binary codes or the sets of excited non-input signals in* s *and* s' *are identical.*

We now introduce semi-modular SGs with input choices which are SGs used in asynchronous circuit design.

Definition 2 (Semi-modularity with input choices) *An SG is* semi-modular with input choices *if and only if* $(\forall t_1 \in T_O)$ *and* $(\forall t_2 \in T)$ *and* $(\forall s \in S)$*, we have*

$$s[t_1\rangle \text{ and } s[t_2\rangle \Rightarrow \exists s' \in S : s \xrightarrow{t_1 t_2} s' \text{ and } s \xrightarrow{t_2 t_1} s'.$$

It means that transitions of non-input signals can never be disabled. Only input transitions can be disabled by other input transitions. This decision is assumed to be correctly managed by the environment and hence creates no problem since only a circuit for non-input signals has to be synthesized. The SG of Figure 1 is such a one.

Semi-modular SGs with input choices can be classified into either *distributive SGs* or *non-distributive SGs*. This classification is based on the notion of detonant states.

Definition 3 (Detonant) [17] *A state* w *is* detonant *with respect to a non-input signal* a *if and only if there exists a pair of states* u *and* v *that directly follow* w *(i.e.,* $w \rightarrow u, w \rightarrow v$*) such that* a *is stable in state* w *and is excited in states* u *and* v*.*

Definition 4 (Distributivity) *A semi-modular SG with input choices is* distributive *with respect to a non-input signal* a *if and only if there are no detonant states with respect to* a*.*

The SG of Figure 1 is not distributive because both states 0^*0^*0 and 1^*1^*1 are detonant with respect to c. In order to derive a correspondence between signal transitions and states in the SG, different regions are defined as follows.

Definition 5 (Excitation region) [17] *An* excitation region *of signal* a *in SG is a maximal connected set of states in which* a *has the same value and is excited.*

The excitation region corresponding to transition $*a_i$ is denoted as $ER(*a_i)$. Note that there can be several excitation regions for a corresponding to multiple transitions of a. An excitation region corresponding to a "$+a$" transition is called *up-excitation region*. Similarly an excitation region corresponding to a "$-a$" transition is called *down-excitation region*.

Definition 6 (Quiescent region) [1] *A* quiescent region *of signal a in SG is the maximal connected set of states reachable from some $ER(*a)$ in which a has the same value and is stable.*

The quiescent region corresponding to transition $*a_i$ is denoted as $QR(*a_i)$ and is the set of states between $ER(*a_i)$ and $ER(*a_{i+1})$, where $*a_{i+1}$ denotes the next transition of signal a after $*a_i$.

Figure 1 illustrates excitation regions and quiescent regions of signal c in the example SG.

Property 1 (Output Trapping) *In a semi-modular SG with input choices, once we enter an excitation region $ER(*a)$ of a non-input signal a, we can only leave it by firing $*a$.*

The proof follows from the fact that the SG is semi-modular with input choices, which states that output transitions cannot be disabled.

Definition 7 (Trigger region) *A* trigger region $TR(*a)$ *is a minimal connected set of states in $ER(*a)$ such that once we enter it, we can only leave it by firing $*a$.*

A trigger region is illustrated in Figure 2.

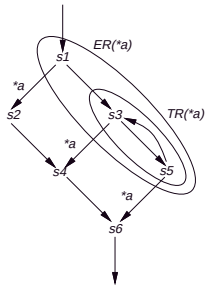

Figure 2: Trigger region

Property 2 (Trigger region reachability) *From any state of an $ER(*a)$, a trigger region is always reachable.*

The proof follows from definitions of excitation and trigger regions and from the output trapping property.

IV N-SHOT ARCHITECTURE

A Overview

In this section, we will first intuitively describe the N-SHOT implementation structure, how it operates, the delay assumptions, and how an SG is implemented at the circuit-level using this architecture. Then in subsequent sections, we will provide in more details our theory and approach.

In this work, we assume a *pure delay* model where gates have pure delays. That is, a pulse of any length that occurs on a gate input can propagate to the gate output. In Figure 3, the N-SHOT implementation structure is presented. It can be used to implement in a hazard-free way almost any semi-modular SG's with input choices, including non-distributive ones, that satisfy the CSC property. Our strategy is to use this implementation structure to implement each non-input signal in the SG specification. The architecture consists of the following:

1. a *Set* sum-of-product (SOP) logic network and a *Reset* SOP network for implementing the *up-transitions* and the *down-transitions* of a non-input signal, respectively — these SOP networks may be minimized using any conventional multi-output two-level minimizers, without special concern for hazards, including the sharing of product terms (AND-gates) between different functions;
2. a new asynchronous storage element called a MHS flip-flop [11] for restoring the primary non-input signals of the specification[1];
3. a "built-in" acknowledgement scheme using two AND-gates and a negligible local delay compensation.

The MHS flip-flop behaves functionally the same way as a C-element. However, there are two differences. First, the MHS flip-flop is dual-rail encoded. Second, it is designed to be electrically robust to small pulses. The latter difference is important because we permit the SOP logic networks to be hazardous, which means small pulses due to hazards may be produced (this will be addressed in details in the subsequent sections).

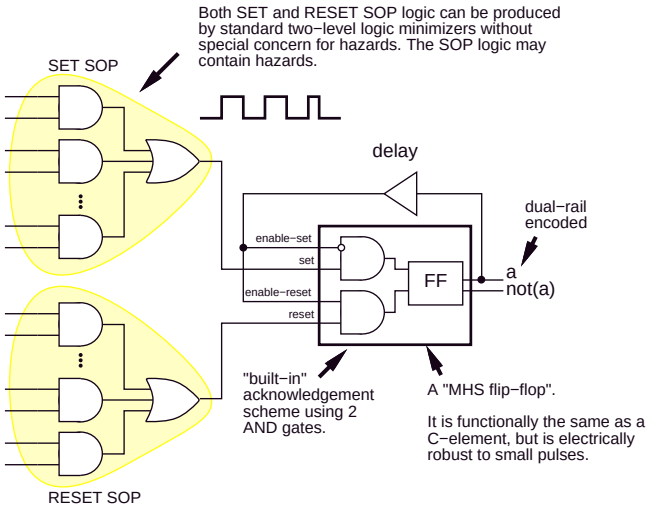

Figure 3: N-SHOT architecture

The *Set* logic network for a non-input signal a is derived as follows:

1. identify all excitation and quiescent regions for a;
2. take the union of all *up-excitation regions*, $\bigcup ER(+a_i)$, and regard it as the *on-set* F of the set function;
3. take the union of all *up-quiescent regions*, $\bigcup QR(+a_i)$, and regard it as the *don't-care-set* D; add all *unreachable states* to the don't-care-set D;
4. take the union of all *down-excitation and down-quiescent regions*, $\bigcup ER(-a_i) \cup \bigcup QR(-a_i)$, and regard it as the *off-set* R.
5. use any multi-output conventional two-level minimizer (e.g. [13]) to produce an optimal sum-of-product implementation from (F, D, R); the don't-care-set D may be used freely.

The *Reset* logic network for a can be derived analogously. With the above procedure for deriving the SOP networks, the set and

[1] M for "M"aster RS latch, H for "H"azard filter and S for "S"lave RS latch.

$s \in S$	SET	RESET	mode
$s \in ER(+a)$	1	0	$+a$
$s \in QR(+a)$	$*$	0	$a = 1$
$s \in ER(-a)$	0	1	$-a$
$s \in QR(-a)$	0	$*$	$a = 0$
unreachable s	$*$	$*$	memory

Table 1: Correspondence between SG regions and operation modes of MHS flip-flop.

reset logic functions of a non-input signal a are in fact characterized by its excitation and quiescent regions in the SG. This correspondence is summarized in Table 1, where $*$ denotes the don't-care value. In the set mode where $+a$ must be fired, $ER(+a)$ is traversed. Some set-SOP cubes are excited and may produce a hazardous response. Then $+a$ is fired and $QR(+a)$ is reached. In the quiescent mode where $a = 1$, $QR(+a)$ is traversed. When $QR(+a)$ is left, $ER(-a)$ is reached. In the reset mode where $-a$ must be fired, $ER(-a)$ is traversed. Some reset-SOP cubes are excited and may produce a hazardous response. Then $-a$ is fired and $QR(-a)$ is reached. In the quiescent mode where $a = 0$, $QR(-a)$ is traversed. When $QR(-a)$ is left, a next $ER(+a)$ is reached. Thus we close a cycle on the operation of the MHS flip-flop relative to the SG traversal.

In our logic derivation procedure, the SOP logic produced requires only AND-gates and OR-gates. Since the flip-flop is dual-rail encoded, inverters are not needed for inverting non-input signals. If all input signals are also presented in dual-rail form, then no inverter at all is needed. Otherwise we assume that either AND-gates with input inversions are available as basic gates[2]. This assumption is also made in the other basic gate implementation methods such as [1, 4]. Under this assumption, the SOP logic never produces 0-1-0 static combinational logic hazards.

In terms of delay assumption, we assume that all gates and wires inside the architecture, including the SOP networks, can have arbitrary delays. Internal wire forks are not required to be isochronic. External I/O wires can also have arbitrary delays. However, I/O signals that are distributed to multiple destinations must have negligible skews. These delay assumptions are weaker than those assumed by speed-independent implementation methods that require all wire forks, both *internal* and external wires, to have negligible skews at multiple-fanout points. In terms of environment assumption, we assume the environment can react immediately, or when it likes, as long as it is enabled to do so in accordance with the SG specification. We do not impose any delay constraints on the environment such as a fundamental mode operation [20].

Since we use conventional logic minimization methods to produce the SOP logic networks, and the gates and wires can have arbitrary delays, the SOP networks may produce hazards that are manifested as streams of pulses. (This is depicted in Figure 3.) As pulse streams are used to excite the flip-flop, two problems must be solved:

- Any pulse stream while traversing one $ER(*a) \cup QR(*a)$ must always be converted into one single $*a$ transition at the output of the flip-flop. This is proven in Section B.
- No pulse stream used to fire $+a$ (resp. $-a$) can continue rippling at the input of the MHS flip-flop after firing of $-a$ (resp. $+a$). This is ensured by the *enable-set* (resp. *enable-reset*) signal that is set to *high* only when $-a$ (resp. $+a$) is fired and when the set (resp. reset) SOP has completely settled to 0. This is the purpose of both AND-gates in the acknowledgement scheme and the local delay compensation.
  Concerning this delay line, the delay compensation requirement is always satisfiable. Moreover, the delay value is negligible because it is related to the *difference* of delays between the set and reset SOPs. If the difference is small compared to the delay of the flip-flop itself, which is almost always the case when sum-of-product implementations are used, then *no* delay compensation is required. When delay compensation is required, the delay is inserted in *parallel* to the flip-flop and generally does not add to the critical path of the combinational logic.
  This is explained in Section C.

Although the gates and wires used in our architecture can have arbitrary delays, *bounds* on the delays must be known in order to determine what local delay compensation, if any, is required. As such, our designs in general are neither speed-independent or delay-insensitive. Again, it is important to remark that usually no delay compensation is required. For all the examples tested in Section V, delay compensation was never required.

B Set and Reset Modes

As both modes are symmetric, we analyze only the set mode. In the set mode, up-excitation regions are traversed. It means that the set SOP logic responds with a possible stream of pulses that excites the MHS flip-flop to fire $+a$. This response is illustrated in Figure 4, where τ is the delay response of the MHS flip-flop, $\omega(\omega < \tau)$ is the delay of the used RS latches and the threshold time value of the MHS flip-flop. The MHS flip-flop does not transmit a pulse shorter than ω and for pulses larger than ω the output transition is simply translated forward in time by τ.

In this section we first describe the MHS flip-flop. Then we show that it is immune to short pulse misbehavior and finally we formulate the requirement on the SG in order to guarantee that the MHS flip-flop can always fire any $*a$.

The logic diagram of the MHS flip-flop with the acknowledgement scheme and a custom designed transistor level implementation (without initializations[3]) are shown in Figure 5. The MHS flip-flop behaves as a C-element. In a transistor to transistor count, it is about the same size as a C-element[4]. However a C-element is not immune to short pulse misbehavior. The MHS flip-flop is composed of three distinct parts that provide hazard filtering in two stages. Its parts and functions are:

1. The master RS latch converts a pulse into an analog voltage.
2. The hazard filter produces hazard-free up-transitions on its output signals (first filtering stage). Some simulation is given in Figure 6 to illustrate the response of the MHS flip-flop to hazardous inputs. We observe first a hazard-free up-transition of signal *slave-set* and a hazardous down-transition of signal *slave-reset*. Then we observe a hazard-free up-transition of signal *slave-reset* and a hazardous down-transition of signal *slave-set*. The filter is in fact two degenerated inverters. The master latch and the filter are the basis of a mutual exclusion element [14, 7]. The latter

[2] If AND-gates with input inversions are not available as basic gates, then we assume the input inversions are implemented with separate inverters whose delays obey some constraints given (and assumed) in [4].

[3] Initialization of the MHS flip-flop is also generally required. This is addressed in Section F.

[4] The actual physical layout of the MHS flip-flop is comparable in physical size to a C-element, even though the transistor counts are not exactly the same.

is used in the context of arbitration (to solve output non-persistencies). It is designed in order to avoid the transmission of metastable states. An analysis on metastable states can be found in [3]. Although we have a different context (semi-modular SGs with input choices do not capture output non-persistencies), we profit from some characteristics of the mutual exclusion element in order to build the first filtering stage of the MHS flip-flop.

3. The slave RS latch eliminates the hazardous down-transitions from the filter (second filtering stage).

A detailed characterization of the behavior of the MHS flip-flop at the electrical level can be found in [10] (Section 4.7.1). There we deeply analyze the principles behind the design of of this new flip-flop, we characterize the response through the MHS flip-flop, when an input pulse of width v (either $v >=$ or $v <$ threshold value) is applied to the master RS latch and we examine how the flip-flop is correctly set and reset.

We now formulate a requirement in order to guarantee that the MHS flip-flop can always fire any $*a$.

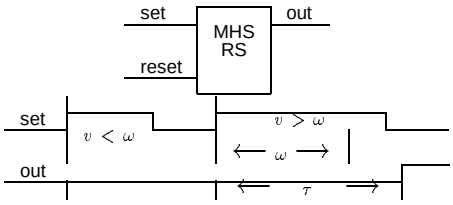

Figure 4: MHS flip-flop response

Requirement 1 (Trigger requirement) *For every non-input signal a implemented with the* MHS *flip-flop and for any transition $*a$ specified in the SG, a pulse must exist that can cause it to fire $*a$.*

Property 3 (Stream to single transition conversion) *If the trigger requirement is satisfied, the* MHS *flip-flop transforms an SOP pulse stream into a single transition during the traversal of the corresponding excitation region.*

Definition 8 (Trigger cube) *A* trigger cube *is a cube from either the set or the reset SOP that completely covers a trigger region.*

Theorem 1 (SG trigger requirement) *The trigger requirement is satisfied if and only if for any non-input signal, there corresponds a trigger cube with each trigger region.*

Proof: $\Rightarrow$ By contradiction. Suppose that in the implementation two cubes are needed to entirely cover some trigger region. Then inside this trigger region we can move from one cube to another one and a stream of pulses can be generated. Since we cannot predict the speed at which those cubes are traversed, those pulses can all be shorter than ω and none of them can cause the MHS flip-flop to fire. Then we may enter a deadlock situation and violate the trigger requirement.
$\Leftarrow$ Suppose that we are traversing $ER(+a)$. Then from the trigger region reachability property, we will always reach some trigger region. Once this trigger region is reached, the corresponding trigger cube switches ON and can only switch OFF after $+a$ is fired, because all reachable states inside the trigger region are covered by the same cube. Thus this trigger cube guarantees that the trigger requirement is satisfied. □

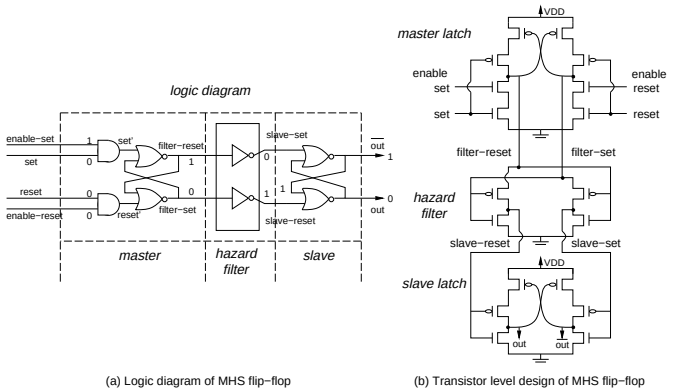

Figure 5: MHS flip-flop

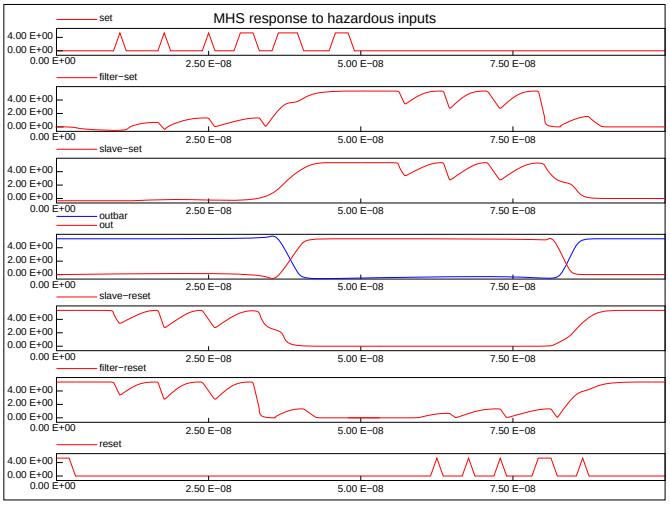

Figure 6: Response of the MHS flip-flop to hazardous inputs

C Quiescent Mode

We now analyze the quiescent mode. We have to make sure that all pulses produced during traversal of $ER(+a_1) \cup QR(+a_1)$ do not go across the traversal of $ER(-a_2)$. If they go across $ER(-a_2)$ and reach $QR(-a_2)$ where $-a$ is already fired, one of those pulses can cause a to misfire during the traversal of $QR(-a_2)$. This trespassing pulse problem is solved using an acknowledgement scheme and some local delay compensation. To derive the delay requirement, let us examine the mechanism behind this acknowledgement scheme (see again Figure 3):

1. During the traversal of $ER(+a_1) \cup QR(+a_1)$, the set SOP produces a pulse stream.
2. When $ER(-a_2)$ is entered, two things happen in parallel:
   - The set SOP is either settling or starts to settle to 0. In the worst case, it takes t_{set0_w} which is the worst case time propagation through the set SOP (2 gates).
   - The reset SOP begins to produce a pulse stream. In the best case $-a_2$ is fired after $t_{res1_f} + t_{mhs-}$, where t_{res1_f} is the fast case time propagation through the reset SOP and t_{mhs-} is the response time of the MHS flip-flop to produce a $-a$.
   - After the firing of $-a_2$, signal *enable-set* becomes 1 again after t_{del}, the value of the delay line. We have to guarantee that the set SOP is completely stabilized

to 0 when this occurs. It implies that

$$t_{del} \geq t_{set0_w} - t_{res1_f} - t_{mhs-}.$$

3. When $ER(+a_3)$ is entered, since the MHS flip-flop is symmetric, we also have to guarantee

$$t_{del} \geq t_{res0_w} - t_{set1_f} - t_{mhs+}.$$

Hence the delay requirement is:

$$t_{del} \geq \text{MAX}\{t_{set0_w} - t_{res1_f} - t_{mhs-}, t_{res0_w} - t_{set1_f} - t_{mhs+}\}. \quad (1)$$

It can always be satisfied and the delay line becomes unnecessary when $MAX \leq 0$. In general t_{del} will be on the order of a gate or less.

A complete analysis of overhead times on critical paths is done in [10] (Section 4.9). Since the delay line is placed in parallel with both set SOP and reset SOP, it is out of their critical paths. Its overhead effect *if any* is indirect.

As we are dealing with transmission of signals through delay lines, it is important to make sure that the signal is really transmitted. Suppose that $+a$ is produced at the output of the MHS flip-flop. Then a is kept stable to 1 because the enabling of the MHS flip-flop to fire $-a$ can only occur after the delay line changes its value from 0 to 1. Hence when $-a$ is produced, it means that $a = 1$ has already been safely translated through the delay line. That is, we have a causal chain of events in the acknowledgement scheme of the N-SHOT architecture that automatically provides the required stability of the signal exciting the delay line.

D Summary of Principles

To summarize, the principle idea behind the N-SHOT architecture is to use the MHS flip-flop as a set-reset element whose set (reset) input must be driven in those states in which the corresponding output is 0 (1), but is excited to 1 (0). It may also be driven (don't care) in those states in which the corresponding output is already 1 (0) and is not excited to 0 (1). Spikes resulting from hazards in the combinational circuits that generate the set and reset inputs are prevented from causing trouble at the output of the MHS by :

1. being input to a master flip-flop whose output is not coupled to the following stage until metastability in the master has resolved. This resolution step makes use of a filter circuitry that is widely used in mutual exclusion elements [14, 7] to eliminate metastability problems. This avoids having weak input pulses from producing any output unless they trigger the master to a changed state.
2. gating off set inputs until a delay time after the output has become 0, and gating off reset inputs until a delay time after the output has become 1. The delay time for the set input is provided largely by the combinational delay of the reset function, and vice versa. A local delay compensation, as shown in Figure 3, parallel to the critical path is inserted when the difference of delay between the set and reset functions are not negligible.

E Hazard-Freeness and Synthesis Procedure

Theorem 2 *A semi-modular SG with input choices satisfying CSC admits a hazard-free implementation within the* N-SHOT *architecture if and only if it satisfies the trigger requirement and the delay requirement.*

The proof follows from the previous analysis on the set and reset mode and on the quiescent mode.

Definition 9 (Single Traversal SG) *A* single traversal SG *is an SG for which any trigger region contains only one state.*

Corollary 1 *Any single traversal semi-modular SG with input choices satisfying CSC always admits an optimal hazard-free implementation within the* N-SHOT *architecture.*

Proof: A single traversal SG always satisfies the trigger requirement and hence a hazard-free implementation always exists. Any minimization technique, such as ESPRESSO-exact can be used to generate a two-level implementation for both set and reset, since we put no constraints on it. □

In Figure 7, both a single traversal SG and a non-single traversal SG are shown. A non-single traversal SG can occur when free running signals like clocks are used in the specifications. Observe that this non-single traversal SG however satisfies the trigger requirement.

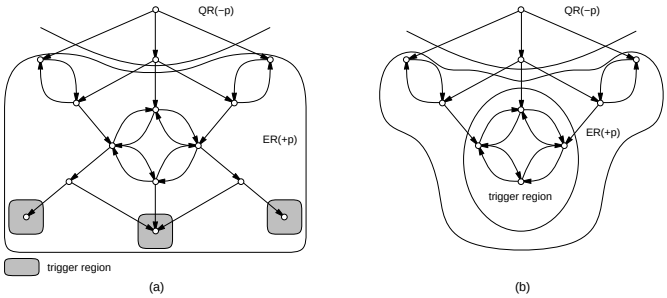

Figure 7: (a) Single traversal SG. (b) Non single traversal SG

Hence the synthesis procedure for any semi-modular SG with input choices satisfying CSC and the trigger requirement is straightforward and is described as follows: 1. Derive an optimal set SOP and an optimal reset SOP using any logic minimizer. If the SG is not single traversal, ensure that a trigger cube corresponds with each trigger region[5]. 2. Map these SOP logic into the N-SHOT architecture. 3. Determine the delay value.

F Initialization of the MHS *flip-flop*

Initialization of the MHS flip-flop is also generally required. One trivial solution is to provide a "reset" product term at one output of the master RS latch in the MHS flip-flop. A short analysis of the final set and reset SOPs can detect when this reset is unnecessary.

For any non-input signal a implemented with a MHS flip-flop, consider the initial state $s_0 \in S$. The reset is only needed in both situations:

1. $s_0 \in QR(+a)$ and $set\_a(s_0) = 0$ in which case the MHS flip-flop must be reset to 1.
2. $s_0 \in QR(-a)$ and $reset\_a(s_0) = 0$ in which case the MHS flip-flop must be reset to 0.

In the other situations the MHS flip-flop is always automatically initialized to either 1 if $s_0 \in ER(+a) \cup QR(+a)$ or 0 if $s_0 \in ER(-a) \cup QR(-a)$.

V Experimental Results

In our view, the most important contribution of this work is a new method that guarantees a hazard-free gate-level circuit for a very broad class of SGs, including non-distributive specifications, that can make use of conventional highly-tuned two-level logic minimization methods (as in the synchronous case) without the usual complexities and difficulties of ensuring hazard-freeness during logic minimization, by only requiring the SG to satisfy the complete state coding property, which is the minimal requirement on SGs necessary to derive unambiguously consistent logic.

We have implemented our method and tested it on a large suite of benchmarks to verify that the solutions produced are of high quality. In particular, the techniques described in this paper have been implemented in the ASSASSIN compiler [21]. In

[5] Again all the examples tested in Section V were single traversal.

Circuit	states	SIS area / del	SYN area / del	ASSASSIN area / del
chu133	24	352 / 5.2	232 / 4.8	256 / 4.8
chu150	26	232 / 7.0	240 / 4.8	240 / 4.8
chu172	12	104 / 1.6	152 / 3.6	120 / 2.4
converta	18	432 / 6.8	496 / 6.0	488 / 4.8
ebergen	18	280 / 5.6	344 / 4.8	312 / 4.8
full	16	224 / 5.2	240 / 4.8	240 / 4.8
hazard	12	296 / 6.6	256 / 4.8	232 / 4.8
hybridf	80	274 / 6.6	352 / 4.8	336 / 4.8
pe-send-ifc	117	1232 / 12.2	1832 / 6.0	1408 / 6.0
qr42	18	280 / 5.6	344 / 4.8	312 / 4.8
vbe10b	256	1008 / 10.0	800 / 4.8	744 / 4.8
vbe5b	24	272 / 4.2	240 / 3.6	240 / 3.6
wrdatab	216	824 / 4.8	840 / 4.8	760 / 4.8
sbuf-send-ctl	27	408 / 5.2	696 / 4.8	320 / 3.6
pr-rcv-ifc	65	1176 / 9.8	1640 / 6.0	1144 / 4.8
master-read	2108	1016 / 6.4	880 / 4.8	824 / 4.8
read-write	315	740 / 7.6	(2)	608 / 6
tsbmsi	1023	(4)	960 / 4.8	928 / 4.8
tsbmsiBRK	4729	(4)	(3)	1648 / 4.8
pmcm1	26	(1)	(1)	304 / 4.8
pmcm2	13	(1)	(1)	160 / 3.6
combuf1	32	(1)	(1)	480 / 4.8
combuf2	24	(1)	(1)	456 / 4.8
sing2dual-inp	65	(1)	(1)	386 / 4.8
sing2dual-out	204	(1)	(1)	648 / 3.6

(1) : Non-distributive SG
(2) : Must add state signals (not handled in version 2.3)
(3) : can be handled with the latest version
(4) : Input file in SG format

Table 2: Experimental results

our implementation, we've used the ESPRESSO minimizer from SIS [15] to produce the sum-of-product implementation[6].

We have applied our implementation to generate hazard-free implementations for two sets of benchmarks. The first set of benchmarks taken from [5, 1] are all distributive specifications. These benchmarks are given as SGs that have already been transformed to satisfy the CSC property. For this class of SGs, we have available to us two tools developed by Lavagno [5] and Beerel [1] for comparisons. The results are reported in the first part of Table 2. The obtained circuits for all three methods were realized using the SIS library. The area and performance estimates were derived using this library. We followed the same strategy chosen in [1] to make area / performance comparisons between the different methods[7].

Concerning the implementations obtained from the SYN tool, version 2.3, of Beerel, 2-level implementations for sets and resets were almost always generated. This explains why results are often similar with those produced by ASSASSIN. Differences are explained because some optimizations present in SYN are not yet implemented in ASSASSIN. For *read-write* and *tsbmiBRK*, SYN version 2.3 either requires the insertion new state signals (which is not handled in SYN version 2.3) or generates memory space problem. Concerning the implementations obtained from the SIS tool of Lavagno, delay lines must be inserted to solve hazards, implying overhead in both area and delay. Both *tsbmsi* and *tsbmsiBRK* are given in SG format and hence cannot currently be handled by the SIS tool. On several examples, our method produced significantly better results. Consider for example the benchmarks *pe-send-ifc*, *wrdatab*, *sbuf-send-ctl* and *pr-rcv-ifc*. On these circuits, our approach produced noticeably smaller circuits than SYN which required extra internal hardware to ensure proper acknowledgement. On the same circuits, and others, our method generally produced faster circuits than SIS which required delay insertions that lengthen the critical path to ensure the circuit is hazard-free.

The second set of benchmarks are *non-distributive* specifications taken from actual industrial designs. The circuits *pmcm1*, *pmcm2*, *combuf1*, and *combuf2* are interface circuits from a mobile terminal design [12]. The circuits *sing2dual-inp* and *sing2dual-out* are interface circuits for performing switchable single-rail to dual-rail conversion. These circuits are needed for the design of an asynchronous DCC decoder [16, 19]. We have carefully simulated these industrial designs at the gate-level using VERILOG and at the transistor-level using SPICE. Results are reported in the second part of Table 2. For these non-distributive designs, no comparison is currently possible.

Finally, it is important to remark that, in all of the examples tested, delay compensation in the N-SHOT architecture was *never* required.

[6] We've used the ESPRESSO command from insider SIS that performs two-level minimization heuristically. Improved results can still be obtained by using the ESPRESSO-EXACT minimizer [13] instead.

[7] See Section 5.1 of [1] for details how gate areas and delays are measured and how the comparisons were made between SYN and SIS.

REFERENCES

[1] P. A. Beerel and T. H. Meng. Automatic gate-level synthesis of speed-independent circuits. In *ICCAD*, November 1992.

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

[3] T. Jackson and A. Albicki. Analysis of metastable operation in D latches. *IEEE Trans. on Circuits and Systems*, 36(11):1392–1404, November 1989.

[4] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *DAC-94*, June 1994.

[5] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *DAC-91*, June 1991.

[6] B. Lin, C. Ykman-Couvreur, and P. Vanbekbergen, A General State Graph Transformation Framework for Asynchronous Synthesis, In *EuroDAC-94*, September 1994.

[7] A.J. Martin. Formal program transformations for vlsi circuit synthesis. In E.W. Dijkstra, editor, *Formal Development of Programs and Proofs.* Addison-Wesley, Reading, MA, 1990.

[8] S.M. Nowick and B. Coates. UCLOCK: automated design of high-performance unclocked state machines. In *ICCD-94*, October 1994.

[9] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T. P. Fang. "Q-modules: Internally clocked delay-insensitive modules". *IEEE Trans. on Computers*, Vol.37-9, September 1988.

[10] M. H. Sawasaki. *General Hazard-Free Synthesis of Asynchronous Circuits.* PhD thesis, Katholieke Universiteit Leuven, February 1994.

[11] M. H. Sawasaki, C. Ykman-Couvreur, B. Lin, and H. De Man. Optimized synchronous logic synthesis mapped into hazard free asynchronous circuits. *US Patent Filling Application*, December 1993.

[12] L. Philips, I. Bolsens, B. Vanhoof, J. Vanhoof, and H. De Man. Silicon Integration of Digital User-end Mobile Communication Systems. In *IEEE International Conference on Communications*, May 1993.

[13] Richard Rudell. Logic synthesis for VLSI design. Technical Report UCB / ERL M89 / 49, Berkeley, 1989.

[14] C. L. Seitz. System Timing. In *Introduction to VLSI Systems, C.A. Mead and L.A. Conway, editors.* Addison-Wesley, Chapter 7, 1980.

[15] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *ICCD-92*, October 1992.

[16] K. van Berkel. Private communications. December, 1993.

[17] V. Varshavsky, M. Kishinevsky, V. Marakhovsky, V. Peschansky, L.Rosenblum, A. Taubin, and B. Tzirlin. Self-Timed Control of Concurrent Processes. Kluwer Academic Publishers, 1990.

[18] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man, A Generalised State Assignment Theory for Transformations on Signal Transition Graphs, *Journal on VLSI Signal Processing*, Kluwer Academic Press. February 1994.

[19] S. Vercauteren. Interface design for switchable single-rail to dual-rail conversion. EXACT internal report, IMEC, May 1994.

[20] K. Y. Yun, D. L. Dill, S. M. Nowick. Synthesis of 3D asynchronous state machines. In *ICCD-92*, October 1992.

[21] ASSASSIN: A Synthesis System for Asynchronous Control Circuits. User and Tutorial Manual. IMEC, September, 1994.