Synthesis of Hazard-free Customized CMOS Complex-Gate Networks Under Multiple-Input Changes

Prabhakar Kudva[†]
IBM T.J. Watson Research Center
Yorktown Heights

Ganesh Gopalakrishnan[‡] Hans Jacobson
Department of Computer Science
University of Utah

Steven M. Nowick[§]
Department of Computer Science
Columbia University

Abstract

This paper addresses the problem of realizing hazard-free single-output Boolean functions through a network of *customized complex CMOS gates* tailored to a given asynchronous controller specification. A customized CMOS gate network can either be a single CMOS gate or a multilevel network of CMOS gates. It is shown that hazard-free requirements for such networks are less restrictive than for simple gate networks. Analysis and efficient synthesis methods to generate such networks under a multiple-input change assumption (MIC) will be presented.

1 Introduction

This paper addresses the problem of realizing single-output Boolean functions through a network of *customized CMOS gates* for asynchronous controllers. A customized CMOS gate network can be either a single CMOS gate or a multilevel network of CMOS gates, where each CMOS gate is tailored to give the most efficient implementation for a given specification. This differs from the approach where one chooses standard gates from a library (such as AND-OR, MUX, AOI, etc.) to implement the required Boolean function in a hazard-free manner [7, 12, 5]. Techniques will be presented to synthesize such networks without hazards, under multiple-input change (MIC) transitions.

Customized CMOS gate implementations have been successfully used to design a large number of *burst-mode* asynchronous controllers [3, 13]. However, previous methods do not present systematic models and synthesis algorithms to take advantage of the particular hazard properties of these circuits. There are several reasons for considering customized CMOS complex-gate based circuits. As VLSI feature size decreases and wire delays become significant, customized CMOS complex-gates can provide more efficient controller implementations compared to standard-cell place and route tools. Also, the recent availability of better layout synthesis techniques that can automatically generate layouts for arbitrary transistor networks makes customized complex-gate based controllers a more viable alternative. Finally, we provide methods to derive complex-gate networks which relax some of the synthesis constraints needed for hazard-free simple-gate implementations.

Currently, there are two main approaches to deriving hazard-free logic gate networks for asynchronous circuits. The first is a *function region* approach. In this method, one tries to find a hazard-free network for a single output Boolean function by taking into account the on-set and the off-set of the function, with respect to a specified set of multiple-input changes. The second approach deals with finding the *excitation regions* of the function. In this method, the regions of the Boolean space where the output is *enabled to change* are identified as "set" and "reset" functions. These functions are implemented and are used to control the switching of a state holding element such as a C-element or RS latch.

Various techniques for hazard-free logic minimization have been proposed for the function region approach. An exact hazard-free two-level logic minimization algorithm, based on a modified Quine-McCluskey method, is given in [7]. Hazard non-increasing transformations and algorithms for *multilevel* optimization of gate-level logic have been given in [4, 14]. A BDD-based method [5] which targets multilevel multiplexor-based networks has been developed. *Technology-mapping* techniques to perform hazard-non-increasing mapping of two level AND-OR networks into complex gate networks from a standard cell library have been given in [12]. Other technology mapping techniques have implemented Boolean functions as single gate hazard-free CMOS complex gate circuits [13, 3]. However, no systematic procedure to derive such CMOS gates has been outlined, which includes precise hazard-free requirements for these gates.

For the class of methods that use the excitation regions, single CMOS complex-gate circuits, called Generalized C elements [6], have been used as target implementations. These techniques usually rely on the use of state holding elements on the output of the gate.

The contribution of this paper is to address the problem of deriving hazard-free customized CMOS realizations for asynchronous controllers under multiple-input changes, using the function region approach. This problem is encountered during the synthesis of burst-mode circuits [8, 15] and is a general problem in asynchronous synthesis. In particular, we present a style of CMOS gate design, called *SOP/SOP form*, that *reduces* the constraints in hazard-free synthesis of single CMOS complex gates. Second, we present a generalization of this technique to multilevel networks. This technique allows efficient solutions to a large class of asynchronous specifications. These techniques allow designers the flexibility to perform hazard-free mapping tailored to customized complex-gates, instead of being confined to a standard library.

In Section 2, we will introduce some basic terminology. Section 3 describes a technique to derive single CMOS complex gates. We will present techniques that address multilevel synthesis of such complex gate circuits in Section 4. Section 5 will provide conclusions and the open problems.

[†] This research was done when the first author was a graduate student at the University of Utah and was supported in part by University of Utah Research Fellowship.

[‡] Supported in part by NSF Award MIP 9215878

[§] This research was supported by an NSF CAREER Award MIP-9501880 and by an Alfred P. Sloan Research Fellowship.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

2 Terminology

In this section, we first present definitions relating to pass transistor and CMOS logic gates. We will then briefly describe some terminology on hazards.

2.1 Pass Transistor Networks

A model for pass transistor logic has been developed in [9, 10, 11, 12]. We describe and extend the model presented in these works for single CMOS gates.

Definition 1 *A* **pass transistor** *is a MOS transistor operated as a switch, where the transistor drain (source) is connected to the signal to be passed along, the transistor gate is connected to the control input, and the output signal is taken from the transistor source (drain).*

At this point, we do not distinguish between a single N-type or P-type MOS transistor and a pass "gate" composed of complementary pair of transistors connected by complementary control variables.

Definition 2 *A* **pass network** *is an interconnection of pass transistors which realizes a particular switching function $f(X)$, where $X = \{x_1, x_2, \ldots, x_n\}$ is the set of inputs to the function.*

Definition 3 *A* **branch** *of a pass network implementing the switching function $f(X)$ is a series connection of pass transistors where the drain or source of the transistor at one end of the series is connected to an input source selected from the set $\{0, 1, x_i, x_i'\}$.*

Definition 4 *A* **pass variable** *is an input to a branch of the pass network. A pass variable may be chosen from the set $\{0, 1, x_i, x_i'\}$.*

Definition 5 *A* **control variable** *is an input to the gate of a transistor in the pass network. When the control variable has a value equivalent to a logic "1", the transistor conducts.*

Definition 6 *A* **pass implicant** *is a Boolean switching function which denotes the function of a branch of a pass network, and includes information about both the pass variable and the switching function. A pass implicant is denoted $C_i[P_i]$, where C_i is a product term composed of control variables which control the pass transistors in that branch, and P_i is the pass variable which will get passed to the output if the product term is true.*

A pass implicant is similar to an implicant in Boolean algebra, except that instead of passing a constant "1" if the implicant is true, the value of the pass variable is passed if the implicant is true.

Definition 7 *A* **pass function**, *$F_p : B^n \rightarrow \{0, 1, Z\}$ is a sum of pass implicants.*

Definition 8 *A* **CMOS gate** *consists of a P transistor pass network with only one pass variable "1" and an N transistor pass network with only one pass variable "0". The outputs of the two networks are connected together to form the output of the CMOS gate. For a CMOS gate which is to implement a function F, the P pass network implements the pass function F and the N pass network implements the pass function $\overline{F}$.*

2.2 Hazards

There are two basic classes of combinational hazards: *function* and *logic* hazards. Function hazards are a property of the logic function, whereas logic hazards are purely a property of the implementation. Within the class of logic hazards, there are single-input change (SIC) hazards and multiple-input-change (MIC) hazards. Additionally, each class of hazards (function and logic) includes both *static* and *dynamic* hazards. Further definitions and details regarding hazard modeling can be found in [14]. In this paper we will consider MIC logic hazards, i.e., we will assume that the given Boolean function free of function hazards for each specified input change.

3 Hazard-free Single CMOS gates

CMOS complex gate networks can be implemented in many different ways. The standard technique to implement the functions F and $\overline{F}$ is to obtain a sum of products for one pass network (p or n) and the dual of this network then becomes the complementary product of sums network.

We will present a more interesting realization in terms of hazard behavior, where *both F and $\overline{F}$ are implemented as sum of products networks*, referred to as an *SOP/SOP form* of complex gates. An example SOP/SOP complex gate implementation is shown in Figure 1. Function F is implemented using p-channel stacks, and Function $\overline{F}$ is implemented using n-channel stacks.

The delay model assumed in this work is that of unbounded gate and wire delays, as in previous approaches [2, 7, 5, 4]. This is a conservative model, which assumes that inputs in an MIC transition can arrive at any time and in any order, and that gates and wires have unknown delay. However, our model is limited by one timing constraint: on the time period for which the capacitance on a CMOS gate output holds its charge when there is no conducting path to power or ground rails through the p or n transistor networks (only leakage occurs). This time period is assumed to be much larger than the *duration of any static hazard*. This requirement is quite reasonable, since the time is related to the maximum difference in arrival of a variable and its complement to different stacks in the gate.

3.1 SOP/SOP Realization

We will first examine the hazard behavior of the SOP/SOP form of realization for CMOS complex gates. In order to do this, we will examine both SIC and MIC static and dynamic transitions on a case-by-case basis.

Case 1: Static Transitions. *For static transitions a SOP/SOP complex gate circuit is hazard-free at the output.* Both SIC and MIC static hazards occur when a given static transition causes a change from one cube of the cover to another, causing a brief period when the transistor network is not conducting. Consider F and $\overline{F}$ to be on-set and off-set covers respectively implemented as p and n transistor networks in a complex gate. It has been shown in [14] that a sum-of-products implementation of the on-set F does not have any $0 \rightarrow 0$ hazards.[1] Similarly, $\overline{F}$ does not have any $1 \rightarrow 1$ hazards. This means that in a sum of products form, a static transition over function F is always outside the cover of $\overline{F}$ and vice versa. As a result, for a SOP/SOP form of complex gates, even if the transistor network of F has a static hazard (that is, a brief moment when no p stack is conducting), the transistor

[1] Note that, throughout this paper, we assume that no product contains both a variable and its complement, otherwise additional hazards are possible [14].

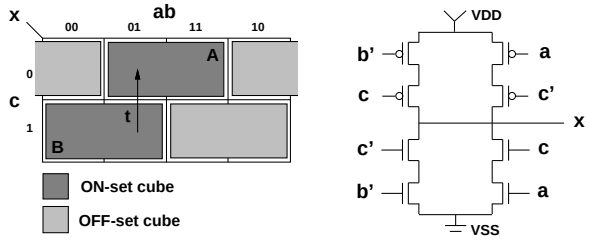

Figure 1: K-map and static hazard-free SOP/SOP complex gate

network of $\overline{F}$ remains off (that is, no n stack will conduct during the transition). Therefore, *the output capacitance of a SOP/SOP complex gate holds its current charge for the duration of a static hazard* (we have no conducting path to either power or ground), and the hazard is not seen at the output.

As an example, consider the Karnaugh map and complex-gate implementation in Figure 1. Transition t, from $abc : 011 \rightarrow 010$, is a static $1 \rightarrow 1$ transition. The on-set is implemented in the p-transistor pass network, using cubes $A = bc'$ and $B = a'c$. This cover is hazardous for a simple-gate AND-OR network. During transition t, the AND-gate for B goes low and the AND-gate for A goes high. If the AND-gate for A is slower than the AND-gate for B, the OR-gate output will glitch. In contrast, the single complex-gate network is hazard-free. Although the p-transistor stacks for A and B can briefly be off at the same time (when c goes low), *no* n-transistor pass network will conduct during the transition. As a result, the output will hold its current charge. Therefore, there is no need to avoid static hazards during synthesis of F and $\overline{F}$.

Case 2: Dynamic Transitions. For the case of SIC transitions, it has been shown in [14] that a dynamic SIC hazard cannot occur (assuming no product contains both a variable and its complement). Since F and $\overline{F}$ are in two-level AND-OR form, no hazards will occur in the complex-gate output in this case. For the case of MIC transitions, though, we will have to make the p and n pass networks hazard-free for dynamic transitions. Otherwise, even in the SOP/SOP form, both the p network and the n network may have dynamic hazards, creating a hazard at the output of the complex gate.

3.2 Algorithm For SOP/SOP Realizations

Our hazard-free algorithm for SOP/SOP complex-gate realizations is similar to an existing algorithm for hazard-free two-level simple-gate networks [7]. The key difference is that our new algorithm uses *fewer constraints*: we can ignore hazards due to static transitions in the SOP/SOP realization.

We first summarize the *Make-sets* algorithm given in [7], which is the first step in the hazard-free two-level minimization of AND-OR simple-gate implementations. We then present a modified version, *Complex-Make-sets*, to handle complex-gate circuits. Finally, we describe the remaining steps which are common to both the two-level algorithm and the complex-gate algorithm. *Make-sets* is first in a series of three steps in exact hazard-free two-level logic minimization. This algorithm finds the required set cubes (*req-set*), the *off-set* cubes and the privileged set (*priv-set*) cubes. The **required set** contains the **required cubes** of the function, for the given MIC transitions. Each $1 \rightarrow 1$ static transition, and the maximal on-set cubes during a dynamic transition, are required cubes. To insure no static hazards for a given $1 \rightarrow 1$ transition, its required cube must be contained in some implicant in the cover. To insure no static hazard for each static sub-transition in a dynamic $1 \rightarrow 0$ or $0 \rightarrow 1$ transition (*i.e.*, the portion of the transition where the output remains at $1 \rightarrow 1$), each maximal on-set cube must be contained in some implicant of the cover. These constraints insure that the output will not glitch during a given static transition. (Further details are provided in [7].)

The priv-set is the set of **privileged cubes**, corresponding to the $1 \rightarrow 0$ and $0 \rightarrow 1$ transition cubes. Each dynamic transition is regarded as a privileged cube, with a specified start point. These cubes are used to prevent dynamic hazards. In particular, no implicant in the cover may intersect a privileged cube *unless* it includes the start point of the privileged cube. If an implicant intersects a privileged cube but does not contain its start point, it has an *illegal intersection*, and may not be included in the cover. Intuitively, such an implicant may turn on, then off, during a dynamic transition, resulting in an output glitch. These conditions are justified in detail in [7].

Implicants which have no illegal intersections are called *dynamic hazard-free implicants (DHF-implicants). Only DHF-implicants may be included in a hazard-free cover.* "Maximal" DHF-implicants, which cannot be expanded further, are called *dynamic hazard-free prime implicants (DHFPI).* In our modified algorithm *Complex-Make-sets*, we follow the same steps as *Make-sets*, but with one key difference: we do *not* generate required cubes for $1 \rightarrow 1$ static transitions. The reason is that, for complex-gate realizations, there are no static $1 \rightarrow 1$ hazard-free requirements. Instead, to insure that the on-set of the function is still covered, we simply add the on-set *minterms* (*not* cubes) into the required set which are not already present in the required set. In summary, the required set generated by Complex-Make-sets consists of (i) all the required cubes associated with dynamic transitions, and (ii) the on-set minterms that are not already covered by other required cubes.

The steps after the generation of sets are common to both the complex-gate algorithm and two-level algorithm, and are summarized below. First derive the DHF prime implicants based on the req- and off-sets as well as the privileged cubes. A *unate covering problem* is then formulated: the problem is to cover all the required cubes by a minimum set of dhf-prime implicants. If all of the required cubes cannot be covered by the dhf implicants, a solution does not exist.

Note that the unate covering problem in our complex-gate algorithm is less restrictive: required cubes for static $1 \rightarrow 1$ transitions need not be covered. In fact, there are problems which have no 2-level hazard-free solution, but where a complex-gate solution exists. For instance, the example used in [7] to demonstrate the absence of a solution for hazard-free AND-OR implementation, has a solution in the SOP/SOP form of complex-gate implementation.

To determine the practical applicability of the SOP/SOP form of complex gates, circuits from the state machine benchmarks [15] were used for a comparison. We applied both the CMOS complex-gate method and the two-level simple-gate methods, and used the Cadence schematic entry system and the LAS [1] layout synthesiser. Results are shown in Table 1 in the form of critical path delays under same input slopes and output load.

As expected, we found that the complex gate circuits performed better in some cases and were slower in others. The performance of the complex gates is dependent on the height of the transistor stacks. As the height of the stack increases the complex-gate implementation performance decrease. This prompted us to investigate the problem of using SOP/SOP form for multilevel complex-gate circuits.

Circuit Name	*Output*	*C. Gate*	*Std.Gate*
chu-ad-opt	dr	1.1 ns	1.5 ns
	lr	1.75	1.4
van-bek-ad-opt	dr	0.9	1.3
	zr	0.92	1.34
	lr	1.2	0.93
sendr-done	DoneS	1.3	1.36
sbuf-read-ctl	Ack	1.3	1.36
	RamRdSbuf	0.97	1.53
q42	a4	0.94	1.56
	r2	1.32	1.61

Table 1: Complex-gate Versus Standard Gate Implementations

4 Multi-level Implementations

4.1 Background and Overview

Several approaches have been used for multilevel hazard-free logic synthesis.

In [2], a technique was presented to derive single-output multilevel AND-OR gate implementations. The algorithm assumes a fully-specified function and attempts to eliminate hazards even for unspecified transitions, leading to inefficient implementations. Our method takes a similar approach but removes hazards only for a given set of transitions.

A method using BDDs that target multilevel multiplexer based circuits is presented in [5]. The multiplexers in this method are assumed to be hazard-free. Work in [12] targets multilevel hazard-free circuits, starting from a hazard-free two-level circuit. In this method a hazard-free two-level function is decomposed into base functions using De Morgan's theorem and associative laws and then partitioned into cones which are mapped to library elements based on associated hazards. This work could be extended to use customized complex gates instead. However, since it is based on an already existing AND-OR function, it cannot take advantage of the static hazard robust behavior of the SOP/SOP form and thus cannot give solutions to a larger class of problems.

We will therefore present a new technique which is an extension of work in [2], but which deals with the special hazard requirements of SOP/SOP complex-gates. The procedure is presented in two steps. First, we will give a model for the multilevel complex gates we target. We will then outline our decomposition algorithm.

4.2 CMOS Multilevel Networks

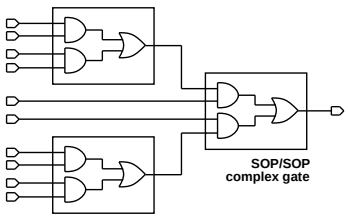

Figure 2: Multilevel SOP/SOP complex gates

The procedure outlined in the rest of this section, assumes that each gate is implemented in the SOP/SOP form discussed in the last section. A multilevel network of CMOS complex gates is defined as a single output network of multiple levels of complex gates, where the control variable for each transistor of the p and n pass networks is either an input literal or the output of another CMOS complex gate.

Consider the goal of implementing a Boolean function under a given set of MIC transitions as a single complex gate. Consider the p pass network to be implemented (the arguments for the n network are symmetric). We attempt to derive the SOP form (series/parallel) network with only input literals as control variables to the transistors. If such a solution cannot be found, we attempt to find a solution where some of the transistors have control variables which are the output of separately implemented complex gates in the SOP/SOP form. This procedure yields alternating levels of AND and OR gates starting from the output and recursively derives implementations for smaller functions until input literals are reached, as can be seen in Figure 2. For synthesis of such multilevel circuits, one must keep in mind that the target complex gates do not require static hazard covers; therefore, we still take advantage of the static hazard-free nature of the SOP/SOP form. We will use this model to derive our complex gates taking the p pass network and n pass network separately. Note, though, that hazards now may occur due to the interaction of separate complex-gates in the network. These issues are addressed below.

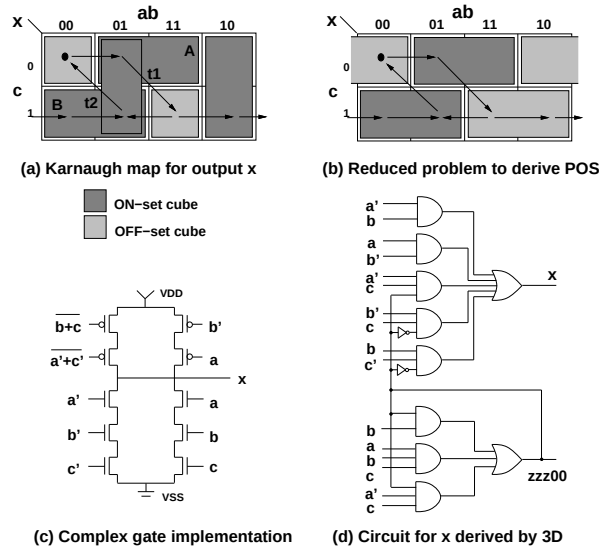

Figure 3: Multilevel SOP/SOP complex gate example

We will illustrate the method with the example shown in Figure 3. Consider the problem of generating a hazard-free single output function for the output x of a burst mode state machine represented by the Karnaugh map in Figure 3(a). In this case, x is a simple combinational function, which must be implemented without logic hazards. Specified input transitions are given in the figure. The required cubes for dynamic transition $t1$ are bc' ("A") and $a'b$, and the required cubes for dynamic transition $t2$ are $a'c$ ("B") and $a'b$.

This covering problem has no hazard-free two-level solution. Each required cube must be covered by some dhf-prime implicant. Required cube "A" is covered only by itself (bc'), which illegally intersects dynamic transition $t2$. Similarly, required cube "B" is covered only by itself ($a'c$), which illegally intersects dynamic transition $t1$. Therefore, no dhf-prime exists to cover "A" and "B".

Burst-mode sequential synthesis tools [8, 15] avoids this prob-

lem at an earlier point in synthesis: during state minimization. By using careful constraints on state merger, these methods produce Boolean functions for which a hazard-free solution exists. That is, a feedback variable would be added, making the circuit sequential rather than combinational.

We now attempt to derive multilevel combinational logic for the output x. First we derive all required cubes and dynamic hazard-free prime implicants (DHFPI). The DHFPIs are $a'b$ and ab' for the on-set. All other implicants have illegal intersections. The two required cubes that cannot be covered in the on-set are bc' ("A") and $a'c$ ("B"). The DHFPIs for the off-set are $a'b'c'$ and abc. The cubes (for both on and off-set) that can be covered are now implemented as a SOP/SOP complex gate.

We will now try to derive a product of sums implementation of the union of the uncovered cubes in the on-set. The reduced problem for this is shown in Figure 3(b). A hazard-free cover for this on-set is $(b+c)(a'+c')$. This POS cover is connected to the SOP/SOP complex gate by a series p stack of transistors as can be seen in Figure 3(c). Since DHFPI $a'b$ is already covered by this hazard-free cover we can remove it from the final cover for x. $(b+c)$ and $(a'+c')$ are separately implemented as complex gates. Note that the static hazard for transition abc:$101 \rightarrow 001$ does not manifest in the SOP/SOP multilevel implementation. Also, no state variable is needed.

Figure 3(d) shows the result as generated by the 3D [15] synthesis tool, which uses *hfmin* [7] to produce a hazard-free two-level AND-OR gate implementation. A state variable has been added to eliminate the hazard problem in this case.

An algorithm for deriving multilevel complex gates is given in Figure 4. The top level algorithm is $Derive\_CMOS\_Multi$, which calls the recursive procedure $Derive\_Multi$ and then derives complex gate implementations from the cover returned by $Derive\_Multi$. The function $Derive\_Multi$ is first discussed. Initially it is assumed that one attempts a two-level sum of products solution. Since we are dealing with alternating *levels* of sum of products and product of sums implementations, our algorithm works slightly differently for each of the levels. We derive the sum of products and the product of sums alternately. Therefore the algorithm first starts with trying to find a sum of product solution. In the absence of static hazards, such a solution may not exist when a required cube(s) for one dynamic transition is a (are) stray cube(s) for another. We will refer to such cubes as *conflicting* cubes. Sets of conflicting cubes are formed. For example if required cubes A and B are conflicting and similarly B and C, the set (A, B, C) is considered the maximal set of conflicting required cubes. For each maximal set of conflicting required cubes, it attempts a product of sums solution, and then again recursively for the remaining cubes attempts a sum of products and so on. Since we have an algorithm targeted to find the minimal sum of products implementation, we also convert the problem of finding the product of sums for a function F to the problem of finding the sum of products for $\overline{F}$ and then using De Morgan's law (which has been shown to be hazard-preserving) [14] to obtain F. The inputs to the algorithm are: the level (indicating whether it is a sum of products or product of sums problem), the set of input transitions, the on-set of the function for which a hazard-free implementation is to be derived.

4.3 Algorithm

We will now describe all the steps outlined in the algorithm. **Step 1** is to derive the req, off and priv sets as described in the last section using algorithm Complex-make-sets if one is targeting the sum of products (say for the first level or any odd numbered level); otherwise, the algorithm Make-sets is used. DHF prime implicants are then derived in **Step 2**. The set of DHF prime implicants is called DHFPI. The set of required cubes due to static transitions if any (in even levels) are recorded as $rset_{static}$. In **Step 3**, the covering problem is attempted. The required cubes that remain not covered by the DHFPI are noted, we will call this set $rset_u$. Note that the only required cubes that remain not covered by the DHFPIs are due to conflicting dynamic hazard transitions during a sum of products minimization problem, i.e., a required cube of one transition becomes a stray cube of one or more other dynamic transitions and vice versa. In the case of a product of sums implementations, the remaining required cubes could also be due to a static hazard requirement as well as due to conflicting dynamic transitions.

Derive_Multi(Level, Set T of input Transitions, On-set)
Step 1. if Level = odd
  Complex-make-sets(T, On-set)
 else
  Make-sets (T, On-set)
Step 2. Derive DHFPI set, req cubes $rset$, $rset_{static}$
Step 3. $Cover = MinCover(DHFPI, rset)$
  if $Cover$ return $Cover$
Step 4. For ($rset_u$ = req cubes not covered by DHFPI)
  if (any $rset_{static} \in rset_u$) goto 6
  $rset_u^i$ = set of conflicting cubes in $rset_u$
  For each $rset_u^i$
   on-set (on_u^i) = $\cup$ $minterms$ in $rset_u^i$
   if (on_u^i already attempted) goto 6
   if Level = odd
    if (c_u^i = Derive_Multi(Level+1, T , off-set(on_u^i)))
     $Cover = Cover \cup ApplyDeMorgan(\overline{c_u^i})$
    else goto 6
   else
    if (c_u^i = Derive_Multi(Level+1, T, off-set(on_u^i)))
     $Cover = Cover \cup c_u^i$
    else goto 6
Step 5. If any $DHFPI_j \in DHFPI$ is covered by a on_u^i
   $DHFPI = DHFPI - DHFPI_j$;
  $Cover = Cover \cup MinCover(DHFPI, rset - rset_u)$;
  return $Cover$;
Step 6. No solution. return NIL
end
Derive_CMOS_Multi(Set T of input Transitions, On-set)
 if (Cover = Derive_Multi(1, T, On-set))
  Partition each AND-OR level starting from output.
  Derive complex gates for partitions
 else
  return NIL
 end

Figure 4: Algorithm

In **Step 4** we find the set $rset_u^i$ of all cubes from $rset_u$ that conflict. The goal then is to derive an implementation for this on-set which is hazard-free for the original set of input transitions. If we are trying to solve the problem for the first level (AND-OR), it is clear that a sum-of-products implementation of $rset_u^i$ will not solve the problem. Instead a hazard-free product of sums implementation of this on-set is attempted. Note that in the algorithm the method to derive this dual implementation is a recursive call. In order to derive a products of sum implementation for an on-set G, the recursive call attempts to derive a sum of products implementation for the off-set $\overline{G}$ and then uses DeMorgan's law on G to obtain the product of sums implementation. A sum of products

problem can be minimized ignoring static transitions since every sum of products function is implemented within a single SOP/SOP CMOS gate. Therefore the product of sums problem (even levels) will use Make-sets in step 1, whereas the sum of products will use Complex-Make-sets in step 1. In **Step 5**, all DHFPI's at that level that are covered by the new cubes are removed. **Step 6** indicates cases where there are no multilevel solutions of this form.

The function $Derive\_CMOS\_Multi$ is the top level function that derives the cover (if there is one) using $Derive\_Multi$ and then partitions each AND-OR level starting from the output. A hazard-free complex gate is then derived for each partition, which may require further application of function $Derive\_Multi$.

We have synthesized many examples using this technique. Layouts have been obtained using the Cadence synthesis tools. For these examples, our method obtained a hazard-free combinational logic solution whereas the 3D tool [15] (using *hfmin* [7]) often had to add several state variables *just in order to prevent logic hazards*. Results for cases where occuring hazards require a multilevel solution for the SOP/SOP complex gate form are given in Table 2. The area required for a hazard-free cover is shown in the $area$ columns and the number of state variables that had to be added to get a solution in the two-level standard gate implementation is shown in the $statevar$ column. The average delay from input event to output response under same input slopes and output load is shown in the $delay$ columns.

Due to the reduced hazard constraints during synthesis, we have obtained encouraging results from our experiments. In our experiments, the area required to get a two-level solution greatly exceeds that of our SOP/SOP complex gate implementation. Half of this area is typically used for added state variables. However, even when the area for these is not included, the complex gate implementation use significantly less area in all examples. A comparison of input-output latency was also made. Due to fewer transistors and the ability to size the transistors very accurately to comply with the size of the output load, we are able to get performance gains of over 50% in many cases. For these examples all complex gate implementations were combinational. These combinational circuits provide an advantage compared to the sequential circuits produced by the 3D method with respect to fundamental mode delay.

Name	S.Gate statevar	C.Gate area	S.Gate area	C.Gate delay	S.Gate delay
bus_tr	2	271	1107	0.30	0.55
run_dp	1	360	1218	0.38	1.35
si_stack	1	177	312	0.20	0.32
sm_stat	2	187	1181	0.31	0.74
sm_dyn	2	222	1109	0.23	0.59
l_cov	2	173	824	0.21	0.58
comp_dp	2	395	1162	0.39	0.70

Table 2: Multilevel Complex-gate versus Standard Gate

5 Conclusions

In this paper, we have presented a technique to synthesize a hazard-free network of customized CMOS complex gates. We have presented a summary of properties and synthesis algorithms for a style of single customized CMOS gate and multilevel CMOS gate networks. The work was motivated by the fact that customized CMOS complex gates could provide flexibility of design, performance and area improvement and solutions to larger classes of problems in hazard-free asynchronous controller synthesis. During the analysis, we have also shown that certain combinational functions which have no solution in the two-level AND-OR implementation form have a solution in our CMOS gate method. Also, our method provided combinational logic solutions in some cases where the two-level method could produce a solution only by adding state variables.

Acknowledgment: The authors would like to thank Al Davis for many helpful discussions.

References

[1] Custom layout/Virtuoso LAS user's manual, cadence design systems, inc., 1992.

[2] BREDESON, J. G. Synthesis of multiple input-change hazard-free combinational switching circuits without feedback. *Int. Journal Electronics 39*, 6 (1975), 615–624.

[3] DAVIS, A., COATES, B., AND STEVENS, K. The Post Office Experience: Designing a Large Asynchronous Chip. In *Proceedings of the 26th Annual Hawaiian International Conference on System Sciences, Volume 1* (Jan. 1993), T. Mudge, V. Milutinovic, and L. Hunter, Eds., pp. 409–418.

[4] KUNG, D. Hazard-non-increasing gate level optimization algorithms. In *International Conference on Computer Aided Design (ICCAD), Santa Clara* (Nov. 1992).

[5] LIN, B., AND DEVADAS, S. Synthesis of hazard-free multi-level implementations under multiple-input changes from binary decision diagrams. In *Proc. International Conf. Computer-Aided Design (ICCAD)* (Nov. 1994).

[6] MARTIN, A. J. Programming in VLSI: From communicating processes to delay-insensitive circuits. In UT Year of Programming Institute on Concurrent Programming (1989), e. C.A.R. Hoare, Ed., Addison-Wesley.

[7] NOWICK, S., AND DILL, D. L. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Transactions on Computer-Aided Design 14*, 8 (Aug. 1995), 986–997.

[8] NOWICK, S. M., AND DILL, D. L. Automatic synthesis of locally-clocked asynchronous state machines. In *Proc. International Conf. Computer-Aided Design (ICCAD)* (Nov. 1991), IEEE Computer Society Press, pp. 318–321.

[9] PEDRON, C., AND STAUFFER, A. Analysis and synthesis of combinational pass transistor circuits. *IEEE Transactions on CAD/CAS 6*, 5 (1988), 727–750.

[10] RADHAKRISHNAN, D., WHITAKER, S., AND MAKI, G. Formal design procedures for pass transistor switching circuits. *IEEE Journal of Solid State Circuits 20*, 2 (1985), 531–536.

[11] SASI, S., AND RADHAKRISHNAN, D. Hazards in cmos circuits. *Int. Journal Electronics 68*, 6 (1990), 967–990.

[12] SIEGEL, P. Automatic technology mapping for asynchronous designs. Tech. rep., Ph.D Thesis, Computer Systems Laboratory, Stanford University, 1993.

[13] STEVENS, K. Ph.d thesis. Tech. rep., Computer Systems Dept, University of Calgary, 1994.

[14] UNGER, S. H. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, 1969.

[15] YUN, K. Y. *Synthesis of asynchronous controllers for heterogeneous systems*. PhD thesis, Stanford University, Aug. 1994.