Removing Hazards in Multi-Level Logic Optimization for Generalized Fundamental-Mode Asynchronous Circuits

Feng Shi
Skyworks Solutions, Inc.
feng.shi@skyworksinc.com

***Abstract*— Unlike traditional synthesis methods for fundamental-mode asynchronous circuits which require dedicated hazard-free algorithms, a multi-level logic optimization algorithm is developed to take advantage of the powerful and mature synchronous synthesis algorithms and technology libraries. The proposed algorithm is based on a hazard analysis method, which not only detects any hazard in an arbitrary circuit structure, but also identifies the cause of the hazard. Then, a hazard removal process is performed on the circuit synthesized using synchronous algorithms to generate a hazard-free circuit. The proposed synthesis algorithm achieves high efficiency by exploiting synchronous optimization algorithms and technology libraries, as demonstrated through the experimental results.**

I. Introduction

Research interest in asynchronous circuits has revitalized because of a number of advantages that they promise over their synchronous counterparts, such as low power, no clock skew, robustness to environmental variations, good modularity, and low electromagnetic interference. Hazard-free logic synthesis algorithms are of great importance to design of asynchronous circuits. Unlike their synchronous counterpart, asynchronous circuits require hazard-free logic in order to operate correctly. Synchronous circuits are protected from such errors, since all the glitches generated before the circuit stabilizes are masked by clock signals, and do not propagate through latches or flip-flips. However, these unwanted glitches increase the energy consumption of the circuit. Therefore, hazard-free logic may also be utilized by synchronous circuits to reduce their power consumption.

Among the set of logic synthesis techniques, this paper focuses on multi-level logic optimization and technology mapping for asynchronous circuits. Traditional logic synthesis methods for synchronous circuits do not guarantee the freedom of hazard conditions, hence, cannot be directly utilized to synthesize asynchronous circuits. Design of fundamental-mode asynchronous circuits, such as burst-mode machines, usually requires dedicated design flows [1]. For instance, hazard-non-increasing transformations [2], [3] are utilized in optimization and technology mapping for multi-level burst-mode machines. However, these design tools are still in infant stage in comparison with powerful and numerous EDA tools for synchronous circuits, which is one of the reasons for the limited adoption of the asynchronous design style by the industry.

In this paper, we propose a multi-level logic optimization and technology mapping method for fundamental-mode asynchronous circuits. Our method builds upon commercially available logic simulation and synthesis tools for synchronous circuits, with minimum add-on to guarantee the hazard freedom. The rest of this paper is organized as follows. In Section II, we introduce the basic aspects of asynchronous technology. In Section III, we review previously proposed logic synthesis methods for fundamental-mode asynchronous circuits. In Section IV, we present a hazard detection method which does not use complicated multi-valued logic. In Section V, we describe a hazard-free logic optimization algorithm which eliminates the hazards in a circuit synthesized by traditional EDA tools for synchronous circuits. In Section VI, we demonstrate the efficiency of the proposed multi-level logic optimization method through the experimental results on a set of example circuits.

II. Background

The fundamental-mode circuit style is one of the popular asynchronous design styles. In this section, we introduce basic definitions and properties of asynchronous circuits and combinational hazards, particularly concentrating on the fundamental-mode circuit style.

A. Classes of Asynchronous Circuits

Asynchronous circuits are divided into two main categories according to their design style, namely *Huffman* and *Muller* circuits. Muller circuits [4] are designed mainly based on signal transition graphs (or Petri Nets) as the specification form. Under the unbounded gate delay model, these circuits are guaranteed to work regardless of gate delays, assuming that wire delays are negligible. Muller circuit design requires explicit knowledge of the behavior protocol allowed by the environment. However, no restrictions are imposed on the order or speed that inputs, outputs, and state signals change, except that they must comply to this protocol. Muller circuits correspond to Speed-Independent circuits.

Huffman circuits [4] are designed using a traditional asynchronous state machine approach. Correctness of Huffman circuits relies on the assumption of "fundamental operation mode", which requires that outputs and state variables stabilize before either new inputs or changed feedback state variables arrive. Thus, delay elements may be required along the feedback paths to prevent state changes from occurring

978-1-4244-2658-4/08/$25.00 ©2008 IEEE

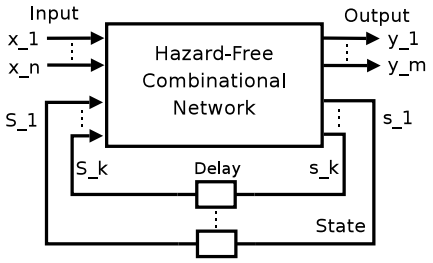


Fig. 1. Block Diagram of A Burst-Mode Machine

too rapidly. Other than that, Huffman circuits are guaranteed to work regardless of gate and wire delays. Contemporary Huffman circuits, such as burst-mode machines [5], [6], allow both single-input changes (SICs) and multiple-input change (MICs). A burst of input changes can occur in any order, and the state variables and output signals remain until all input changes complete. Huffman circuits allowing MICs are also named generalized fundamental-mode circuits. Figure 1 illustrates the block diagram of a burst-mode machine, which is one popular style of the generalized fundamental-mode circuits.

B. Hazards

Hazard analysis is critical in design of asynchronous circuits. A hazard, in the most general sense, is an unwanted glitch on the output of a gate in a circuit. The presence of hazards may cause an asynchronous circuit to operate incorrectly. A hazard may exist in either the combinational or the sequential portion of a circuit. We only consider *combinational* hazards in this paper, since *sequential* hazards are handled in other synthesis steps such as state minimization and encoding. 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.

Alternatively, combinational hazards can be classified into *static* and *dynamic* hazards. Given that a transition is being made on logic f between two points $\mathbf{a}$ and $\mathbf{b}$ in the input space $\{0,1\}^n$, static hazards apply to cases where $f(\mathbf{a}) = f(\mathbf{b})$, and dynamic hazards apply to cases where $f(\mathbf{a}) \neq f(\mathbf{b})$. Single-input changes may cause both classes of hazards if no constraint is put on the structure of the circuit, and multiple-input changes are more likely to do so. We consider both SICs and MICs in this paper, and a SIC can usually be treated as a special case of MIC.

Figure 2 illustrates an example of the static hazards caused by a SIC transition. Fig. 2 (a) shows the truth table that the circuit in Fig. 2 (b) is implementing, with initial and final input states of the given transition circled. Although it is free of function hazards according to the truth table, the circuit may output a static hazard rather than a constant one as specified. Moreover, the example in Figure 3 demonstrates how a MIC transition generates a dynamic hazard. As illustrated in Fig. 3 (a), the ideal circuit should output a falling transition when the inputs switch from $(A, B, C, D) = (0, 1, 1, 1)$ to $(1, 1, 1, 0)$. However, the circuit implemented in Fig. 3 (b) may generate a static hazard on node n_{10}, which then causes a dynamic hazard on the circuit output.

III. Previous Work

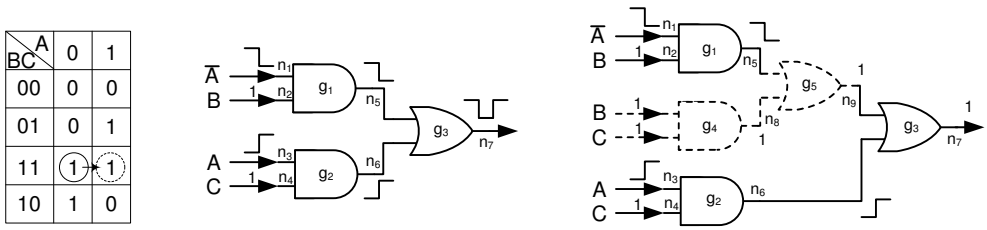
This paper concentrates on multi-level logic optimization and technology mapping techniques for generalized fundamental-mode circuits. Many researchers have studied the problem of synthesis and technology mapping for fundamental-mode circuits. Exact hazard-free two-level logic minimization algorithms have been developed [7], [8]. Furthermore, heuristic hazard-free two-level minimization algorithms have also been developed [9], [10]. Several approaches on synthesizing multi-level hazard-free circuits have also been proposed. One approach starts with hazard-free two-level circuits, and applies hazard-non-increasing transformations to obtain multi-level hazard-free circuits [11], [2]. Another approach synthesizes hazard-free multi-level circuits directly, using binary decision diagrams [12]. Moreover, a number of hazard-free technology mapping algorithms have been developed. Siegel et al [13] proposed an algorithm for generalized fundamental-mode asynchronous circuits by modifying an existing synchronous technology mapper. Beerel et al [14] developed technology mapping techiniques that optimize for average case delay of asynchronous burst-mode control circuits. However, all of the above approaches need to use dedicated hazard-free algorithms for asynchronous circuits, which limits the adoption of asynchronous design style by the industry.

IV. Multi-Level Hazard Analysis

The proposed hazard-free multi-level logic optimization and technology mapping algorithm is based on hazard analysis for fundamental-mode asynchronous circuits. We only need to analyze hazards in combinational logic, since the logic optimization algorithm only transforms the combinational portions of the design, while sequential portions remain untouched. The proposed analysis method directly identifies hazards in multi-level circuits without using complicated multi-valued algebras for asynchronous circuits. Moreover, both static and dynamic hazards, or SIC and MIC hazards, are detected using a uniformed procedure. Several terms need to be defined before the proposed method is formally presented.

A. Linearly Separable Logic Gates

A *gate* is an atomic component of a circuit in the proposed hazard analysis algorithm. A gate is assumed to have no logic hazard, but it can have arbitrary output delay. A gate in the technology library can usually be regarded as an atomic gate, unless it contains logic hazards, and must be replaced with an equivalent subcircuit composed with atomic gates during the hazard analysis. Among various gate functions, Boolean *linearly separable* functions are of particular interest, since they are amicable to hazard analysis. Given a Boolean function $f(x_1, \ldots, x_n) : \mathbb{B}^n \rightarrow \{0, 1\}$, the set of points in $\mathbb{B}^n$ that map to 0 is denoted as $\mathcal{X}_0$, and the set of those map to 1 as $\mathcal{X}_1$. The Boolean function f is linearly separable if



(a) Truth Table (b) Original Circuit (c) Hazard-Free Circuit

Fig. 2. Identifying and Removing a Static Hazard Generated by a SIC Transition

there exists a hyperplane Π in $\mathbb{R}^n$ that strictly separates $\mathcal{X}_0$ from $\mathcal{X}_1$, and $\Pi \cup \{0,1\}^n = \phi$.

Given a logic gate, the control value refers to a certain combination of values assigned on a subset of its inputs, which fully determine the gate output, i.e. the values on the rest of the inputs have no influence on the output value. For instance, a *zero* on any input of an AND gate set the gate output to *zero*, regardless of the values on the other inputs. Therefore, *zero* is the control value of an AND gate. Similarly, *one* is the control value of an OR gate. The definition of the control value can be generalized for an arbitrary gate. Assume that combinational gate g has n inputs and one output y, and it implements Boolean function f_g. If the input vector is $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$, then output $y = f_g(\mathbf{a})$. Give an input i,$i = 1, 2, \ldots, n$, the vector of the rest inputs is defined as $\mathbf{a^{(i)}} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, and the concatenate operation $\circ$ is defined as $\mathbf{a^{(i)}} \circ b = (a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n)$, where $b \in \{0,1\}$. Generally, vector $\mathbf{a^{(i)}} \in \{0,1\}^{n-1}$ is the *control value* for input i of gate g if and only if $f_g(\mathbf{a^{(i)}} \circ 0) = f_g(\mathbf{a^{(i)}} \circ 1)$. The set of all such control values is defined as $C_g^{(i)}$, or the *control set* for input i of gate g. For each input of a given gate, we can derive its control set according to the above definition.

A signal transition on an input of a gate implementing a linearly separable function may only change the gate output in one direction, regardless of the values on the other inputs, as long as they remain stable. For instance, a rising transition on one input of a NAND gate never causes a rising transition on the output, no matter the other input is set to 0 or 1. Formally, we have the following theorem.

Theorem 4.1: If a gate g implements a linearly separable Boolean function $f : \mathbb{B}^n \rightarrow \{0,1\}$, and i is one of its inputs, then $\forall \mathbf{a_1^{(i)}}, \mathbf{a_2^{(i)}} \notin C_g^{(i)}$, $f(\mathbf{a_1^{(i)}} \circ 0) = f(\mathbf{a_2^{(i)}} \circ 0)$, and $f(\mathbf{a_1^{(i)}} \circ 1) = f(\mathbf{a_2^{(i)}} \circ 1)$.

Proof: Since $f(x_1, \ldots, x_n)$ is linearly separable, input set $\mathcal{X}_0$ is separated from $\mathcal{X}_1$ by a hyperplane Π, assumed to satisfy equation $\mathbf{c} \cdot \mathbf{x} = c_1 x_1 + \ldots + c_n x_n = d$, where $c_1, \ldots, c_n, d \in \mathbb{R}$. Because $\mathbf{a_1^{(i)}} \notin C_g^{(i)}$, $f(\mathbf{a_1^{(i)}} \circ 0) \neq f(\mathbf{a_1^{(i)}} \circ 1)$. First consider the case $(\mathbf{a_1^{(i)}} \circ 0) \cdot \mathbf{x} = c_1 a_1 + \ldots + c_{i-1} a_{i-1} + c_{i+1} a_{i+1} + \ldots + c_n a_n > b$, and $(\mathbf{a_1^{(i)}} \circ 1) \cdot \mathbf{x} = c_1 a_1 + \ldots + c_{i-1} a_{i-1} + c_i + c_{i+1} a_{i+1} + \ldots + c_n a_n < b$. Subtract the first inequality from the second one, we get $c_i < 0$. Since $\mathbf{a_2^{(i)}}$ is also not in $C_g^{(i)}$, if $(\mathbf{a_2^{(i)}} \circ 0) \cdot \mathbf{x} < b$ and $(\mathbf{a_2^{(i)}} \circ 1) \cdot \mathbf{x} > b$, similarly we can get $c_i > 0$, which contradicts the previous result. Therefore, it must be true that $(\mathbf{a_2^{(i)}} \circ 0) \cdot \mathbf{x} > b$ and $(\mathbf{a_2^{(i)}} \circ 1) \cdot \mathbf{x} < b$, thus $f(\mathbf{a_1^{(i)}} \circ 0) = f(\mathbf{a_2^{(i)}} \circ 0)$ and $f(\mathbf{a_1^{(i)}} \circ 1) = f(\mathbf{a_2^{(i)}} \circ 1)$. Similarly, we can prove that the above conclusion holds for the case $(\mathbf{a_1^{(i)}} \circ 0) \cdot \mathbf{x} < b$ and $(\mathbf{a_1^{(i)}} \circ 1) \cdot \mathbf{x} > b$. ■

B. Hazard Identification

We use the polarity, a Boolean value, to denote different types of transitions. A signal transition can be represented by a 2-tuple of Boolean values. For instance, a rising transition, i.e. switching from 0 to 1, is denoted as $(0,1)$, while a falling transition as $(1,0)$. The *transition polarity* is formally defined as follows.

Definition 4.2: The transition polarity is a function $p : \{(0,1),(1,0)\} \rightarrow \{0,1\}$, which satisfies $p(0,1) = 1$, and $p(1,0) = 0$.

Specifically, the polarity of a rising transition is one, and that of a falling transition is $zero$. Note that the polarity is only defined for transitions, i.e. it is not defined for either $(1,1)$ or $(0,0)$.

Given an input i of an arbitrary gate, if the other inputs are not set to any of the control values, a transition on input i must generate another transition on the output of the gate. Moreover, if the gate function is linearly separable, the polarity of the output transition is determined by the input transition, regardless of the values on the other inputs, as long as they are not in the control set. Therefore, for each input of a linearly separable logic gate, we use the *gate polarity* to indicate if the gate changes the polarity of the input transition.

Definition 4.3: For a given input i, the polarity of a logic gate g which implements a linearly separable Boolean function f_g is $r_g^{(i)} = f_g(\mathbf{a^{(i)}} \circ 1)$, where $\mathbf{a^{(i)}} \notin C_g^{(i)}$.

For instance, the polarity of OR gate g_3 in Fig. 2 (b) regarding input n_5 is $r_{g_3}^{(n_5)} = 0 \vee 1 = 1$ according to Definition 4.3, which means that the input signal is not inverted by the OR gate. Note that the gate polarity is not defined for a non-linearly separable logic gate, where the polarity of the output transition depends on both the changing input and the other inputs.

In addition, if both the polarity of the input transition and the gate polarity are known, we can reason about the polarity of the output transition. For example, if the polarity of the transition on input n_3 in Fig. 3 (b) is 0, and the polarity of AND gate g_2 regarding input n_3 is 1, the polarity of output transition must be 0, given that the other input n_4 is not

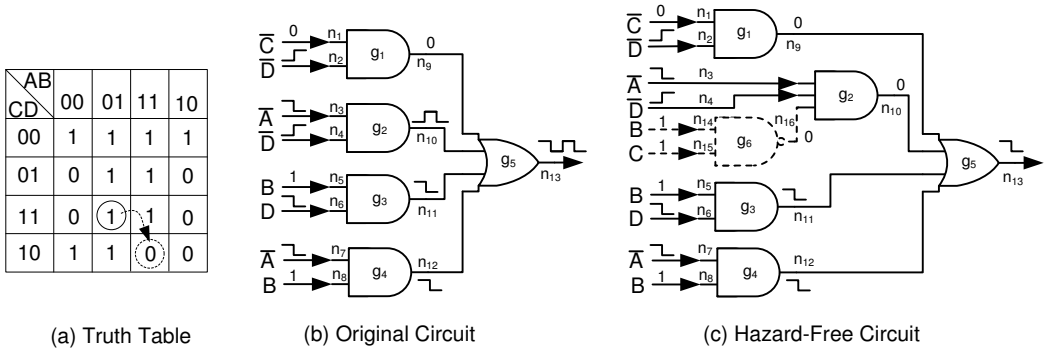


Fig. 3. Identifying and Removing a Dynamic Hazard Generated by a MIC Transition

set to 0, the control value. Formally, we have the following corollary.

Corollary 4.4: If the polarity of the transition on input i of a linearly separable logic gate g is ρ_i, and the other inputs are set to $\mathbf{a^{(i)}} \notin C_g^{(i)}$, then the polarity of the output transition is $\rho_o = r_g^{(i)} \odot \rho_i$, where $r_g^{(i)}$ is the gate polarity regarding input i, and $\odot$ denotes the exclusive-nor operation on Boolean algebra.

The proposed hazard analysis algorithm is based on the above definitions and theorems. Either the stimulus to a logic gate is a single-input change, or a multiple-input change, it can be denoted by two input vectors, i.e. $\mathbf{a}, \mathbf{b} \in \mathbb{B}^n, \mathbf{a} \neq \mathbf{b}$, where $\mathbf{a}$ and $\mathbf{b}$ denote the initial and final values of the inputs, respectively. It denotes a SIC if the Hamming distance between $\mathbf{a}$ and $\mathbf{b}$ equals 1, and a MIC if larger than 1. Moreover, the input pair $\mathbf{a}$ and $\mathbf{b}$ uniquely define a transition space. A transition space, $T[\mathbf{a}, \mathbf{b}]$, is the smallest Boolean subspace which contains $\mathbf{a}$ and $\mathbf{b}$. Under the given input transition, whether a gate may generate a hazard on the output can be determined using the following theorem.

Theorem 4.5: A logic gate g implementing a linearly separable Boolean function f generates a hazard on its output under input change from $\mathbf{a}$ to $\mathbf{b}$ if and only if there exists a transition $t^{(i)}$ on input i, and $T[\mathbf{a^{(i)}}, \mathbf{b^{(i)}}] \not\subseteq C_g^{(i)}$, such that $p(t^{(i)}) \odot r_g^{(i)} \neq f(\mathbf{b})$, where $r_g^{(i)}$ is the gate polarity regarding input i, and $\mathbf{a^{(i)}}, \mathbf{b^{(i)}}$ are the vectors applied on the inputs other than i.

Proof: Since $T[\mathbf{a^{(i)}}, \mathbf{b^{(i)}}] \not\subseteq C_g^{(i)}$, there must exist $\mathbf{c^{(i)}} \in T[\mathbf{a^{(i)}}, \mathbf{b^{(i)}}]$ and $\mathbf{c^{(i)}} \notin C_g^{(i)}$. As described in Section II-A, a fundamental-mode circuit has arbitrary gate and wire delay. Therefore, there must exist a combination of input delays that set the inputs other than i to $\mathbf{x^{(i)}} = \mathbf{c^{(i)}}$ for a certain time during which transition $t^{(i)}$ occurs on input i. As a result, the gate output changes to $p(t^{(i)}) \odot r_g^{(i)} \neq f(\mathbf{b})$, according to Corollary 4.4. However, the output must change to $f(\mathbf{b})$ finally. Therefore, there exists a hazard at the gate output.

Now consider the case $p(t^{(i)}) \odot r_g^{(i)} = f(\mathbf{b})$, for any input i which is applied with transition $t^{(i)}$, and $T[\mathbf{a^{(i)}}, \mathbf{b^{(i)}}] \not\subseteq C_g^{(i)}$. Assume that $t^{(i)} = (v_1, v_2)$, and $\mathbf{x^{(i)}} = \mathbf{c^{(i)}}$ when $t^{(i)}$ occurs. if the gate output has changed to $f(\mathbf{b})$, it must be true that $\mathbf{c^{(i)}} \in C_g^{(i)}$, otherwise $f(c^{(i)} \circ v_1) \neq f(c^{(i)} \circ v_2) = p(t^{(i)}) \odot r_g^{(i)} = f(\mathbf{b})$, which contradicts the assumption that the output has been $f(\mathbf{b})$ already. Therefore, $t^{(i)}$ does not affect the gate output. So $t^{(i)}$ can only cause the output to switch to $f(\mathbf{b})$ when it is still $f(\mathbf{a})$. Therefore, no hazard can be generated if $p(t^{(i)}) \odot r_g^{(i)} = f(\mathbf{b})$. ■

Note that no constraint is placed on the Hamming distance between input vector $\mathbf{a}$ and $\mathbf{b}$, therefore, Theorem 4.5 may be used to detect both SIC and MIC hazards. In addition, it detects both static and dynamic hazards. Moreover, Theorem 4.5 detects not only logic hazards, but also function hazards.

We demonstrate how to detect hazards using Theorem 4.5 through several examples. First, consider gate g_3 in Fig. 2 (b), where input vector (n_5, n_6) switch from $(1, 0)$ to $(0, 1)$. For transition $t^{(n_5)}$ on input n_5, $\mathbf{a^{(n_5)}} = (a_{n_6}) = (0)$ and $\mathbf{b^{(n_5)}} = (b_{n_6}) = (1)$, therefore, $T[\mathbf{a^{(n_5)}}, \mathbf{b^{(n_5)}}] = \{1, 0\} \not\subseteq C_{g3}^{(n_5)} = \{1\}$. Moreover, $p(t^{(n_5)}) \odot r_{g3}^{(n_5)} = 0 \odot 1 = 0 \neq f_{g3}(0, 1) = 1$. Therefore, gate g_3 generates a hazard. The result is consistent with the fact that a two-input OR gate generates a glitch when a rising input transition arrives after a falling input transition, as illustrated in Fig. 2 (b). Another example is gate g_2 in Fig. 3 (b), where input (n_3, n_4) switch from $(1, 0)$ to $(0, 1)$. For transition $t^{(n_4)}$ on input n_4, $\mathbf{a^{(n_4)}} = (a_{n_3}) = (0)$ and $\mathbf{b^{(n_4)}} = (b_{n_3}) = (1)$. As a result, $T[\mathbf{a^{(n_4)}}, \mathbf{b^{(n_4)}}] = \{1, 0\} \not\subseteq C_{g2}^{(n_4)} = \{0\}$. Moreover, $p(t^{(n_4)}) \odot r_{g2}^{(n_4)} = 1 \odot 1 = 1 \neq f_{g2}(0, 1) = 0$. Therefore, a hazard is detected on output n_{10} of gate g_2. Again, the result complies with the fact that a two-input AND gate generates a glitch when a falling input transition arrives after a rising input transition.

V. Hazard-Free Logic Optimization

The theorems in the previous section can be utilized not only to detect hazards in a fundamental-mode asynchronous circuit, but also to build hazard-free logic synthesis algorithms. In the following, we present the hazard removal algorithm.

Theorem 4.5 not only detects a hazard in the circuit, but also identifies the cause of the hazard. The transition $t^{(i)}$ satisfying Theorem 4.5 is named a *hazardous* transition, which in fact causes the hazard. Therefore, the hazard can be easily removed by blocking it.

Theorem 5.1: Under input transition $\mathbf{a} = (a_1, \ldots, a_n)$,$\mathbf{b} = (b_1, \ldots, b_n)$, a hazard generated by a linearly separable logic gate may be removed, if any input i with a hazardous transition is gated by inserting logic, transforming the

implemented Boolean function $f(x_1, \ldots, x_i, \ldots, x_n)$ into $f(x_1, \ldots, x_{i-1}, x_i \wedge \overline{c_{\mathbf{A},\mathbf{B}}}, x_{i+1}, \ldots, x_n)$ when $a_i = 0$, or $f(x_1, \ldots, x_{i-1}, x_i \vee c_{\mathbf{A},\mathbf{B}}, x_{i+1}, \ldots, x_n)$ when $a_i = 1$, where $\mathbf{A}$ and $\mathbf{B}$ are the corresponding initial and final input states of the circuit which contains the gate, respectively, and cube $c_{\mathbf{A},\mathbf{B}} = 1$ if and only if the circuit input $(X_1, \ldots, X_m) \in T[\mathbf{A}, \mathbf{B}]$.

The transformation specified in Theorem 5.1 does not alter the Boolean function at the start and end points of each input transition. When the circuit input $\mathbf{X} \notin T[\mathbf{A}, \mathbf{B}]$, $c_{\mathbf{A},\mathbf{B}} = 0$, therefore, the new function reduces to the original function f. Otherwise, when $\mathbf{X} = (X_1, \ldots, X_m) \in T[\mathbf{A}, \mathbf{B}]$, the new gate function is $f' = f(x_1, \ldots, x_{i-1}, a_i, x_{i+1}, \ldots, x_n)$. First, it is obvious that $f'(\mathbf{a}) = f(\mathbf{a})$. In addition, it must be true that $\mathbf{b^{(i)}} \in C_g^{(i)}$, otherwise, $f(\mathbf{b}) = r_g^{(i)} \odot p(a_i, b_i)$, which contradicts with the fact that transition (a_i, b_i) is a hazardous transition as defined in Theorem 4.5. Therefore, $f(\mathbf{b^{(i)}} \circ a_i) = f(\mathbf{b})$, i.e. $f'(\mathbf{b}) = f(\mathbf{b})$. Thus, if any hazardous transition under input transition $\mathbf{A}$,$\mathbf{B}$ is removed using the method in Theorem 5.1 for any gate which affects the circuit outputs, the circuit is hazard-free for the given transition, while the circuit function is preserved, as long as transition spaces intersect with each other only at start/end points. However, the inserted gating logic may introduce hazards under other input transitions. As a result, it must be analyzed for hazards under other input transitions.

The hazard removal method of Theorem 5.1 can be demonstrated using previous examples. First consider the circuit in Fig. 2 (b). As analyzed in Section IV-B, the circuit has a static hazard on output n_7 , given the input transition from $\mathbf{A} = (A, B, C) = (0, 1, 1)$ to $\mathbf{B} = (1, 1, 1)$, where A,B, and C are the primary inputs. According to Theorem 4.5, the transition on input n_5 is the hazardous transition which causes the hazard. In order to remove the hazard, we use Theorem 5.1 to block it. Since in the initial state n_5 is at 1, we insert the gating logic $g = n_5 \vee c_{\mathbf{A},\mathbf{B}}$, where $c_{\mathbf{A},\mathbf{B}} = B \wedge C$. Therefore, the gating logic is implemented as illustrated in Fig.2 (c), where the inserted gates are shown in dash lines. In the modified circuit, the transition on n_5 is blocked and does not reach gate g_3. As a result, the hazard is removed, and the circuit is hazard-free for the given transition. Note that g_3 and g_5 can be further merged into a three-input OR gate to reduce the total area. The same method can also be used to remove dynamic hazards. In the example circuit of Fig. 3 (b), there is a dynamic hazard at output n_{13} when the inputs switch from $\mathbf{A} = (A, B, C, D) = (0, 1, 1, 1)$ to $\mathbf{B} = (1, 1, 1, 0)$, as discussed in Section IV-B. Since the rising transition on input n_4 of gate g_2 is the hazardous transition, we may block it to remove the hazard. According to Theorem 5.1, we insert gating logic $g = n_4 \wedge \overline{c_{\mathbf{A},\mathbf{B}}}$, where $\overline{c_{\mathbf{A},\mathbf{B}}} = \overline{B \wedge C}$. The inserted logic is also illustrated using dash lines in Fig. 3 (c). As a result, the hazardous transition is blocked by gate g_2, and the primary output n_{13} is hazard free. Note that the gating logic may be further simplified, depending on the particular function of the circuit. In this example, the gating logic can simply be $g = n_4 \wedge \overline{B}$ instead of $n_4 \wedge \overline{B \wedge C}$, without changing the function of the circuit. Therefore, incremental optimization is often performed on the hazard-free circuits obtained through Theorem 5.1 to achieve better results.

The proposed hazard removal algorithm may be utilized to build a synthesis algorithm for fundamental-mode asynchronous circuits based on synchronous synthesis algorithms. First, given the specification of a fundamental-mode asynchronous circuit, hazard-free state minimization and encoding is performed using asynchronous algorithms such as CHASM [15], and the specification for the combinational portion is generated. Then, synchronous EDA tools may be used to synthesize a circuit based on the specified combinational function, but the generated circuit may contain logic hazards. After that, the above hazard analysis and removal algorithm is used to eliminate the logic hazards for any input transition, which finally generates a hazard-free asynchronous circuit. The proposed synthesis method utilizes the hazard analysis and removal algorithm as an incremental step to the synchronous synthesis flow, hence is able to take advantage of the powerful and mature optimization algorithms and technology libraries for synchronous circuits. Therefore, as demonstrated in Section VI, it is very efficient in spite of not being a global optimization algorithm.

VI. Experimental Results

The proposed algorithm is implemented in Tcl, since it is concise and easy to interface with synchronous EDA tools. We experimented the algorithm with a set of example burst-mode machines. First, we used MINIMALIST [1] to encode the states and generate the truth table of the combination portion of each design in PLA format, according to the burst-mode specification of the circuit. Meanwhile, all the input transitions were stored for hazard analysis in the next step. Then the PLA file for the combinational portion was fed into commercial synchronous synthesis tools to generate an optimized multi-level circuit. After that, hazards were analyzed and removed from the circuit, as described in Section V. Finally, the result circuit including the inserted gating logic was verified to be free of hazards. During the synthesis, the circuit was mapped to a 120nm synchronous standard cell library. Note that in the experiments we only used library cells implementing linearly separable Boolean functions, i.e. XOR, XNOR, and MUX gates were excluded from technology mapping. In addition, the used library cells were assumed to be free of logic hazards. We also synthesized these burst-mode machines with MINIMALIST and MLO, a multi-level hazard-free logic optimization tool, to compare with the circuits generated by the proposed method. During logic optimization using MLO, the maximal number of input pins for each gate was set to 4, the same as the library cells we used. For both methods, we optimized for minimal area.

The experimental results for each example circuit are listed in Table I. The name of each circuit is listed in the first column, followed by the number of inputs, outputs, and state bits of the circuit in the second, third, and fourth

Circuit Name	No. of Inputs	No. of Outputs	No. of State bits	MINIMALIST & MLO		Synchronous		Proposed Method		
				No. of Gates	Area	No. of Gates	Area	No. of Gates	Area	Reduction Rate (%)
concur-mixer	3	3	3	19	164.5	11	101.6	11	101.6	38.2
pe-send-ifc	5	3	4	54	529	29	280.6	32	312.1	41.0
martin-q-element	2	2	1	6	52.2	4	38.7	4	38.7	25.9
rf-control	6	5	3	29	258.9	23	193.5	23	193.5	25.3
dme-e	3	3	2	14	121.0	9	79.8	9	79.8	34.0
opt-token-distributor	4	4	5	23	196.0	16	147.6	16	147.6	24.7
tangram-mixer	3	3	1	8	70.2	3	33.9	3	33.9	51.7
it-control	5	7	5	40	401.6	27	256.4	32	302.4	24.7
dram-ctrl	8	6	1	35	333.8	23	196.0	31	275.8	17.4
hp-ir	3	2	1	5	55.6	4	36.3	4	36.3	34.7
Average										31.8

TABLE I
AREA OF SYNTHESIZED CIRCUITS

columns, respectively. The number of gates and the total area of each circuit synthesized by MINIMALIST and MLO are reported in the fifth and sixth columns, respectively. The results related to the proposed method are listed from the seventh column to the eleventh column. The seventh and eighth columns show the number of gates and area of each circuit which was synthesized using synchronous tools and may contain logic hazards. As we expected, these area numbers are considerably smaller than those generated by MINIMALIST and MLO. Then, the hazard analysis and removal algorithm was performed to eliminate hazards from these synchronous versions by inserting gating logic, and the number of gates and area of each result circuit are listed in the ninth and tenth columns, respectively. For a number of small circuits such as *concur-mixer*, their synchronous versions are coincidentally hazard-free, so no hazard removal logic is necessary. Note that decreasing the number of gates may also help remove hazards, since a smaller number of gates in the circuit often means less risk of having hazards, and in the extreme case where the circuit is implemented with one single gate, there is for sure no logic hazard. For larger circuits such as *pe-send-ifc*, logic hazards are detected and removed by inserting gating logic. However, the area of each result circuit is still smaller than that generated by MINIMALIST and MLO, which only use a limited set of hazard-free optimization techniques and standard library cells. The eleventh column lists the area reduction rate achieved by the proposed method for each circuit comparing to MINIMALIST and MLO, and the average reduction rate is 31.8%.

VII. CONCLUSION

A multi-level logic optimization algorithm for generalized fundamental-mode asynchronous circuits has been developed, which first optimizes the circuits ignoring the existence of hazards, and thereafter removes the logic hazards in an incremental step to generate hazard-free circuits. The key component of the algorithm is a hazard analysis and removal algorithm which not only detects a hazard, but also identifies the cause of the hazard, and then removes it. In comparison with traditional synthesis methods for asynchronous circuits, which usually constrain themselves on a limited set of hazard-free and hazard-non-increasing techniques, the proposed method achieves better results by exploiting more powerful CAD tools and technology libraries for synchronous circuits, as demonstrated through experimental results.

REFERENCES

[1] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, B. Lin, and L. Plana, "Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines," Columbia University, NY, Tech. Rep. TR CUCS-020-99, July 1999.

[2] D. S. Kung, "Hazard-non-increasing gate-level optimization algorithms," in *International Conference on Computer Aided Design*, 1992, pp. 631–634.

[3] S. H. Unger, "A building block approach to unclocked systems," in *Proc. Hawaii International Conf. System Sciences*, vol. I. IEEE Computer Society Press, Jan. 1993.

[4] C. J. Myers, *Asynchronous Circuit Design*. New York: John Wiley and Sons, Inc., 2001.

[5] S. M. Nowick and D. L. Dill, "Synthesis of asynchronous state machines using a local clock," in *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, Oct. 1991, pp. 192–197.

[6] S. M. Nowick, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. dissertation, Stanford University, Department of Computer Science, 1993.

[7] S. M. Nowick and D. L. Dill, "Exact two-level minimization of hazard-free logic with multiple-input changes," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 8, pp. 986–997, Aug. 1995.

[8] C. Myers and H. Jacobson, "Efficient exact two-level hazard-free logic minimization," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Mar. 2001, pp. 64–73.

[9] M. Theobald, S. M. Nowick, and T. Wu, "Espresso-HF: A heuristic hazard-free minimizer for two-level logic," in *Proc. ACM/IEEE Design Automation Conference*, 1996.

[10] J. Rutten and M. Berkelaar, "Efficient exact and heuristic minimization of hazard-free logic," in *Proc. International Conf. Computer Design (ICCD)*, Oct. 1998, pp. 152–159.

[11] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, John Wiley & Sons, Inc., 1969.

[12] B. Lin and S. Devadas, "Synthesis of hazard-free multi-level logic under multiple-input changes from binary decision diagrams," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1994, pp. 542–549.

[13] P. Siegel, G. D. Micheli, and D. Dill, "Automatic technology mapping for generalized fundamental-mode asynchronous designs," in *Proc. ACM/IEEE Design Automation Conference*, June 1993, pp. 61–67.

[14] P. A. Beerel, K. Y. Yun, and W. C. Chou, "Optimizing average-case delay in technology mapping of burst-mode circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Mar. 1996.

[15] R. M. Fuhrer, B. Lin, and S. M. Nowick, "Symbolic hazard-free minimization and encoding of asynchronous finite state machines," in *Proc. International Conf. Computer-Aided Design (ICCAD)*. IEEE Computer Society Press, 1995.