Exact Essential-Hazard-Free State Minimization of Incompletely Specified Asynchronous Sequential Machines

Fu-Chiung J. Cheng *Luis Plana*[†]

Columbia University
Department of Computer Science
New York, NY 10027
Tel: 212-939-7070, Fax: 212-666-0140
{cheng, plana}@cs.columbia.edu

CUCS-033-94

December 1994

Abstract

To insure correct dynamic behaviour of asynchronous sequential machines, hazards must be eliminated for they may cause malfunctions of the whole system. However, Hazard-free state minimization has received almost no prior attention in the literature. This paper describes an exact algorithm for essential-hazard-free state minimization of incompletely specified asynchronous sequential machines. Novel techniques for the elimination of apparent and potential essential hazards are proposed and exploited in our algorithm. The algorithm has been implemented and applied to over a dozen asynchronous sequential machines. Results are compared with results of non-essential-hazard-free method SIS. Most of the tested cases can be reduced to essential hazard free flow tables.

Keywords: State minimization, essential hazards, hazard-free synthesis, flow tables, asynchronous sequential circuits.

[†]On leave from Universidad Nacional Experimental Politécnica, Barquisimeto, Venezuela. This author is supported by a grant from Consejo Nacional de Investigaciones Científicas y Tecnológicas, Venezuela.

1 Introduction

Interest in asynchronous sequential circuits is growing due to several potential benefits: avoidance of clock skew, low power consumption, average-case instead of worst-case performance, and automatic adaptation to physical properties among others [LKSV91, MBM89, Mar86, ND91].

The synthesis of asynchronous circuits starts with an asynchronous state machine specification and consists of the following three steps: state minimization, state assignment, and logic minimization. One of the most important aspects of asynchronous design is to guarantee that the circuit implementations are hazard-free.

To synthesize hazard-free asynchronous circuits, the following conditions must be satisfied: First, the specification must be free of sequential hazards. Second, the state assignment must be free of critical races. Finally, the implementation must be free of combinational hazards. The exact hazard-free logic minimization for two-level combinational circuits, which solves the combinational-hazard-free problem, has been proposed by Nowick [ND92]. A unicode single transition time state (USTT) state assignment which solves the critical-race-free assignment problem, was proposed by Tracey [Tra66]. One important and difficult problem in designing hazard-free asynchronous sequential circuits is to guarantee that the specification remains free of essential hazards in every stage of the synthesis process. This paper addresses the problem of essential-hazard-free state minimization.

State minimization is an important step for the synthesis of sequential circuits. Many researchers have worked on this problem [Ung69, HRSJ91, PG93]. However, The existing state minimization methods pay no attention to essential hazards. The reason for this may be due to the belief that state minimization has no impact on the presence of essential hazards in a reduced flow table.

This paper presents an exact algorithm for essential-hazard-free (EHF) state minimization of incompletely specified asynchronous machines. Novel techniques to eliminate potential and apparent essential hazards are presented. The goal of EHF state minimization is, given an incompletely specified normal flow table, to find an EHF minimal closed cover, if such a solution exists.

This work is important for the following reasons: first, no hazard-free asynchronous circuit can be built under unbounded delay assumption (i.e. arbitrary finite gate and wire delays) if there are any essential hazards. Second, an asynchronous circuit implemented from an EHF flow table would be fast and robust since no delays need to be added and no glitches will be generated [Ung68].

This paper is organized as follows: Section 2 gives some basic definitions that simplify the discussion. Section 3 illustrates the essential hazard problems in EHF state minimization. The techniques to eliminate and avoid EHs are also proposed here. Section 4 describes the EHF state minimization algorithm in detail. Section 5 gives experimental results. Section 6 concludes this paper.

2 Definitions

To simplify the discussion, we introduce some basic definitions in this section. These definitions are taken from [PG93, Ung69] with minor modifications.

2.1 State Minimization

The behavior of a sequential machine can be described by a flow table. A flow table is a two-dimensional array where columns correspond to the *input states* and rows correspond to the *internal states*. The entries are ordered pairs representing the next state and the output. The next state in state s and input i is denoted by $N(s,i)$ and the output by $Z(s,i)$. The kth literal of the output is denoted by $Z_k(s,i)$. The pair of current state and current input, (s,i), is called *total state*. Flow table ex1 is shown in Table 1. It has 6 states, 3 inputs and 2 outputs. State i is abbreviated as i if no confusion can be generated.

	$x_1x_2x_3$				
	000	001	011	101	010
1	$\boxed{1}$,00	2,- -	-,- -	-,- -	$\boxed{1}$,10
2	-,- -	$\boxed{2}$,11	3,- -	-,- -	$\boxed{2}$,11
3	$\boxed{3}$,00	-,- -	$\boxed{3}$,00	-,- -	4,- -
4	$\boxed{4}$,00	5,- -	$\boxed{4}$,00	-,- -	$\boxed{4}$,00
5	$\boxed{5}$,00	$\boxed{5}$,00	3,- -	6,- -	$\boxed{5}$,00
6	3,- -	$\boxed{6}$,11	-,- -	$\boxed{6}$,11	-,- -

	1	2	3	4	5
2	X				
3	14 X	24 X			
4	25 X	25 34 X			
5	25 X	X	45	34	
6	13 26 X			34 56 X	35 X

Table 1: Incompletely Specified Flow Table ex1 and its Pair Chart

Two states i and j of a flow table are compatible, denoted $i \sim j$, if and only if for every possible input sequence applicable to i and j, the same computed output sequences are produced. On the other hand, if the output sequences differ, then i and j are incompatible. For example, states 1 and 2 in ex1 are incompatible because the outputs, $Z_2(1,010) = 0$ and $Z_2(2,010) = 1$, differ. States 2 and 6, states 3 and 4, and states 3 and 6 are compatible. In some cases, for two states to be compatible, they require other states to be compatible too. For example, the compatibility of states 3 and 5 depends on states 4 and 5 being compatible. The pair chart in Table 1 shows the compatible states, the conditionally compatible states with their implied compatible pairs, and the incompatible states.

A set of states is a **compatible** if and only if every pair of states in the set are compatible. For example, states 3, 4, and 5 are a compatible (named 345) because states 3 and 4, states 4 and 5, and states 3 and 5 are compatible. Table 2 shows all compatibles in flow table ex1. In a similar fashion, a set of states is an **incompatible** if and only if every pair of states in the set are incompatible.

A compatible C_i **covers** compatible C_j, denoted by $C_i > C_j$, if and only if $C_i \supset C_j$. A **maximal compatible** is a compatible that is not covered by any other compatible. For example, compatible 345 is maximal but compatible 34 is not. Similarly, a **maximal incompatible** is an incompatible that is not covered by any other incompatible. The procedures to construct a pair chart and to obtain the maximal compatibles and maximal incompatibles can be found in [Ung69].

The **closure class** $\Phi(C_i)$ of a compatible C_i is a set of all compatibles implied by C_i such that:

1. each implied compatible has more than one state,

2. no implied compatible is a subset of C_i, and

3. no implied compatible is a subset of any other member of the closure class.

For example, compatible 35 implies compatible 45 and compatible 45 implies compatible 34, so the closure class of compatible 35 is {45, 34}. The closure classes of the compatibles of ex1 are shown in the third column of Table 2.

A compatible C_i is said to be **prime** if there exists no other compatible C_j such that:

1. $C_j > C_i$.

2. $\Phi(C_j) \subseteq \Phi(C_i)$.

For example, compatible 345 is prime but compatible 35 is not, for compatible 35 is covered by compatible 345 and $\Phi(345) = \emptyset \subseteq \Phi(35) = \{45, 34\}$. The forth column of Table 2, labeled PC, shows which compatibles are prime.

	Compatible	Closure Class	PC	EHF-PC
1	$<345>$	$\emptyset$	Yes	Yes
2	$<45>$	$<34>$	No	No
3	$<36>$	$\emptyset$	Yes	Yes
4	$<35>$	$<45><34>$	No	No
5	$<34>$	$\emptyset$	No	No
6	$<26>$	$\emptyset$	Yes	Yes
7	$<6>$	$\emptyset$	No	Yes
8	$<5>$	$\emptyset$	No	No
9	$<4>$	$\emptyset$	No	No
10	$<3>$	$\emptyset$	No	No
11	$<2>$	$\emptyset$	No	Yes
12	$<1>$	$\emptyset$	Yes	Yes

Table 2: Compatibles and Corresponding Closure Classes of ex1

The **extended closure class** $\Phi(\delta)$ of a set of compatibles δ is a set of all compatibles implied by δ such that:

1. each implied compatible has more than one state,

2. no implied compatible is a subset of any member of δ, and

3. no implied compatible is a subset of any other member of the extended closure class.

A set of compatibles δ is **closed** if and only if, for every compatible contained in the set, each implied compatible is also contained in at least one compatible of the set. That is, the extended closure class $\Phi(\delta)$ is empty. For example, the set of compatibles {34, 26} is closed but the set {35, 26} is not because $\Phi(\{34, 26\}) = \emptyset$ but $\Phi(\{35, 25\}) = \{45, 34\}$.

Definition 1 *A set of compatibles δ is a* **minimal closed cover** *if and only if δ satisfies:*

1. covering condition: δ covers all the states of the flow table,

2. closure condition: δ is closed, and

3. minimal condition: δ is minimal.

For example, {1, 26, 34, 5} and {1, 26, 345} are both closed covers but the later is minimal and the former is not.

The goal of *state minimization* is to find a minimal closed cover. The state minimization process [PG93, HRSJ91] usually has the following steps:

- Obtain the prime compatibles (or maximal compatibles).
- Select a set of compatibles from the prime compatibles (or maximal compatibles) which satisfies the covering, closure, and minimal conditions.

The set of compatibles {1, 26, 345} is an optimal solution for state minimization[1]. The reduced flowtable is shown in Table 3.

	$x_1x_2x_3$ 000	001	011	101	010	y_1y_2
a(1)	$\boxed{\text{a}}$,00	b,–	-,- -	-,- -	$\boxed{\text{a}}$,10	11
b(26)	c,- -	$\boxed{\text{b}}$,11	c,- -	$\boxed{\text{b}}$,11	$\boxed{\text{b}}$,11	10
c(345)	$\boxed{\text{c}}$,00	$\boxed{\text{c}}$,00	$\boxed{\text{c}}$,00	b,- -	$\boxed{\text{c}}$,00	00

Table 3: A reduced Flow Table for ex1.

2.2 Essential Hazards

A sequential circuit contains a hazard if, for some input change, there is a set of stray delay values that produces a spurious pulse or glitch in a signal or causes the circuit to enter the wrong stable state.

Sequential hazards are present in the circuit specification. They are called essential hazards to denote that they are an inherent property of the sequential function and not of the particular circuit implementation.

There are two types of essential hazards in asynchronous circuits: transient essential hazards (sometimes referred to as output hazards) and steady state essential hazards. A circuit specification is said to contain a *transient essential hazard* (TEH) if, for some input change, a glitch may appear on an output. A circuit specification contains a *steady state essential hazard* (SSEH) if, for some input change, an undesired change may occur in a state variable and, as a result, the circuit may reach an incorrect stable state.

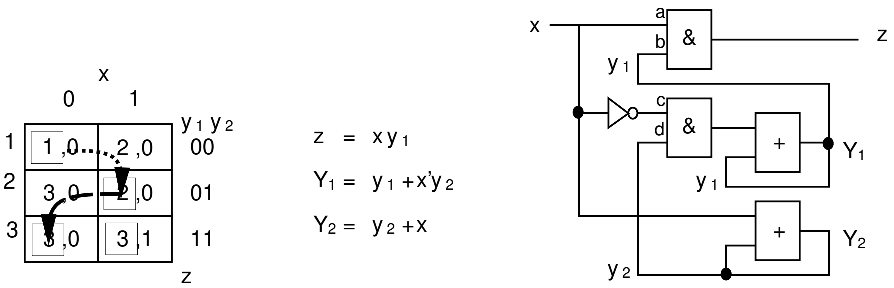
The flow table in Figure 1(a) is used to illustrate the existence of TEHs and SSEHs. The logic expressions and circuit implementation of ex1 are shown in Figures 1(b) and 1(c) respectively.

There is a TEH in the flow table starting in total state (2,1) and x changing from 1 to 0: Initially $xy_1y_2 = 101$; when x turns off, it will cause Y_1 to turn on. Suppose there is big delay between x and a, z may see y_1 change first so it will change from 0 to 1. Eventually x will reset the z output. Thus an output glitch is generated.

There is also a SSEH in the flow table in Figure 1(a) starting in total state (1,0) when x turns on: Initially $xy_1y_2 = 000$; when x turns on, it will cause Y_2 to turn on. Suppose there is big delay between x and c, Y_1 may see y_2 change first and locks itself at 1. Thus the circuit winds up in state 3 instead of state 2.

To synthesize hazard-free circuits, both hazards must be eliminated.

[1]It is a solution for non-essential-hazard-free state minimization but not for EHF state minimization.



(a) Flow Table (b) Logic Expression (c) And–Or Implementation

Figure 1: Examples of TEH and SSEH

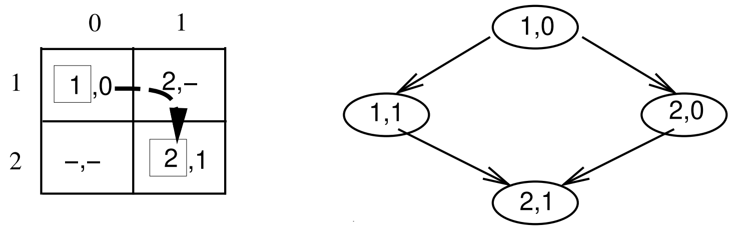
2.3 Transition Trios

Every change in the outputs or state of a sequential circuit is triggered by a transition in an input signal. An input change causes a circuit to move from a total state (i, A) to a total state (j, B). If $i = j$ then the transition is free of essential hazards because there are no intermediate states in the transition. When $i \neq j$, we define a transition trio of a transition as follows:

Definition 2 *Given a transition from a total state (i, A) to a total state (j, B), where $i \neq j$, a* **transition trio** *is a set of three total states, t_1, t_2, and t_3, where t_1 is the starting state, (i, A), t_3 the destination state, (j, B), and t_2 is one of two possible intermediate total states (i, B) and (j, A).*

Each single input change (SIC) transition has two *transition trios*: Let t be a transition trio, $\{(i, A), (k, C), (j, B)\}$.

- t is a **type 1 transition trio** if and only if $i = k \neq j$ and $A \neq C = B$. It represents transitions involving an input change first and then a state change;
- t is a **type 2 transition trio** if and only if $i \neq k = j$ and $A = C \neq B$. It represents transitions involving a state change first and then an input change;



(a) An Input Transition (b) Two Transition Trios

Figure 2: An Input Transition and its Trios.

For example, the transition from (1,0) to (2,1) in Figure 2(a) has two transition trios, {(1,0), (1,1), (2,1)} and {(1,0), (2,0), (2,1)}, shown in Figure 2(b). Transition trios are used to analyze hazards in a flow table.

3 Essential Hazard Analysis

In this section we analyze the possible patterns in a flow table which may cause essential hazards and present novel techniques to eliminate those hazards, whenever this is possible. We consider only SIC, normal flow tables [Ung69]. Don't care output and next state entries are denoted by X.

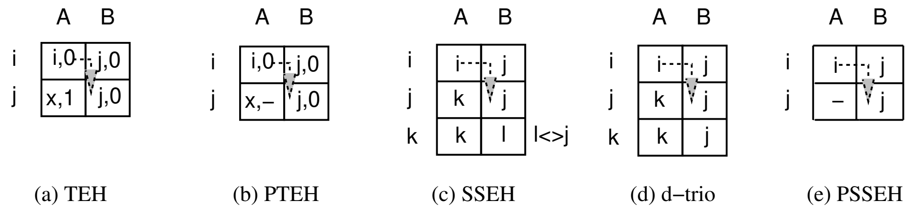


Figure 3: Examples of TEH, PTEH, SSEH, d-trio and PSSEH

3.1 Transient Essential Hazards

Definition 3 *A flow table contains a* **transient essential hazard** *for a transition from total state* (i, A) *to total state* (j, B) *if and only if there exists an output literal* Z_n *such that* $Z_n(i, A) \neq X$, $Z_n(j, B) = Z_n(i, A)$, *and:*

1. $Z_n(i, B) \neq X$ *and* $Z_n(i, B) \neq Z_n(i, A)$, *or*
2. $Z_n(j, A) \neq X$ *and* $Z_n(j, A) \neq Z_n(i, A)$.

The flow table shown in Figure 3(a) illustrates a TEH. There is no way to eliminate a TEH once it exists in a flow table.

The following lemmas, stated without proof, present necessary and sufficient conditions to insure that a flow table is free of TEHs:

Lemma 1 *A transition trio* $\{(i, A), (k, C), (j, B)\}$ *is TEH-free if and only if for each output literal* Z_n, $s = Z_n(i, A)$, $d = Z_n(j, B)$ *and* $t = Z_n(k, C)$ *satisfy one or more of the following conditions:*

1. $s = X$ *or* $d = X$,
2. $s = \overline{d}$,
3. $s = t = d$.

Lemma 2 *A transition* $\{(i, A), (j, B)\}$ *is TEH-free if and only if*

1. $i = j$, *or*
2. $i \neq j$ *and the corresponding type 1 and type 2 transition trios are TEH-free.*

Lemma 3 *A flow table is TEH-free if and only if every transition in the flow table is TEH-free.*

3.2 Potential Transient Essential Hazards

In an *incompletely* specified flow table some patterns that include don't care entries can become TEHs if an inadequate value is assigned to one or more output don't cares. This type of pattern is called a *potential transient essential hazard* (PTEH).

Definition 4 *A flow table contains a* **potential transient essential hazard** *for a transition from total state* (i, A) *to total state* (j, B) *if and only if there exists an output literal* Z_k *such that* $Z_k(i, A) \neq X$, $Z_k(j, B) = Z_k(i, A)$, *and:*
1. $Z_k(i, B) = X$, *or*
2. $Z_k(j, A) = X$.

The flow table shown in Figure 3(b) illustrates a PTEH. This PTEH becomes a TEH if the don't care output in total state (j, A) is specified as 1. If, on the other hand, the don't care output is specified as 0, the PTEH is eliminated.

If the presence of PTEHs is not accounted for during state minimization, a PTEH can become a transient essential hazard. Consider the incompletely specified flow table ex2 in Table 4. The set of maximal compatibles is $\{12, 23\}$. States 2 and 3 are compatible, so they can be merged, (i.e. covered by a single state in the reduced flow table). The reduced flow table is shown in Table 5(a). This reduced table is not unique: rows 1 and 2 can also be merged. The alternative reduced flow table is shown in Table 5(b).

	x_1x_2 00	01	11	10
1	$\boxed{1}$,0	2,0	-,-	$\boxed{1}$,0
2	-,-	$\boxed{2}$,0	3,0	-, -
3	$\boxed{3}$,1	$\boxed{3}$,0	$\boxed{3}$,0	1,0

	1	2
2		
3	23 X	

Table 4: Incompletely Specified Flow Table ex2 and its Pair Chart.

	x_1x_2 00	01	11	10	Y
1	$\boxed{1}$,0	2,0	-,-	$\boxed{1}$,0	0
23	$\boxed{2}$,1	$\boxed{2}$,0	$\boxed{2}$,0	1,0	1

(a) Hazardous FT

	x_1x_2 00	01	11	10
12	$\boxed{1}$,0	$\boxed{1}$,0	3,0	$\boxed{1}$,0
3	$\boxed{3}$,1	$\boxed{3}$,0	$\boxed{3}$,0	1,0

(b) Essential-hazard-free FT

Table 5: Two Reduced Flow Tables of ex2.

Both flow tables in Table 5 have a minimal number of states and have no steady state essential hazard. However, while there are no TEHs in Table 5(b), there is a transient essential hazard in Table 5(a) for the transition from total state (1,00) to total state (2,01).

The TEH appears in the reduced flow table because the output don't care entry in total state (2,00) of ex2 is transformed into a 1 by merging states 2 and 3 during state minimization.

If we constrain the output of the (2,00) entry to 0, then states 2 and 3 are no longer compatible. Minimizing this constrained flow table leads to a single reduced table, the one shown in Table 5(b). As mentioned before, this table has no essential hazards.

According to Lemma 1, in order to eliminate a PTEH from a transition (i.e. to avoid introducing a transient essential hazard), some output functions of the flow table must be constrained: Given a transition trio, $\{(i, A), (k, C), (j, B)\}$, for any output literal Z_n such that $Z_n(i, A) = Z_n(j, B)$ and $Z_n(k, C) = X$, then $Z_n(k, C)$ should be set to $Z_n(i, A)$ to make it TEH-free.

While any particular PTEH can be eliminated by constraining the flow table, it is not always possible to eliminate all PTEHs present in a flow table. If the transition trios associated with two PTEHs involve the same intermediate total state, they might impose contradictory conditions on an output don't care. In this case, only one of the PTEHs can be eliminated.

3.3 Steady State Essential Hazards

Definition 5 *A flow table contains a* **steady state essential hazard** *for the transition from total state (i, A) to total state (j, B) if and only if there exists state k such that $k = N(j, A)$, $k \neq i$, $k \neq j$, and $N(k, B) \neq j$.*

Every SSEH involves three states. The flow table shown in Figure 3(c) illustrates a SSEH (outputs are not shown because they are not relevant). States i (the start state), j (the destination state), and k (the transient state) contribute to the SSEH. Note that k is not specified as a transient state in the transition but it can be reached due to the presence of delays in the circuit. If the hazard manifests, the circuit will go to state l, an incorrect stable state.

The following lemmas, stated without proof, present necessary and sufficient conditions to insure that a flow table is free of SSEHs:

Lemma 4 *A transition trio $\{(i, A), (k, C), (j, B)\}$ is SSEH-free if and only if $N(k, C) = i$ or $N(k, C) = j$.*

An immediate consequence of Lemma 4 is that type 1 transition trios are always SSEH-free.

Lemma 5 *A transition $\{(i, A), (j, B)\}$ is SSEH-free if and only if*
1. $i = j$, or
2. $i \neq j$ and the corresponding type 2 transition trio is SSEH-free.

Lemma 6 *A flow table is SSEH-free if and only if every transition in the flow table is SSEH-free.*

3.4 d-trios

The flow table shown in Figure 3(d) contains a pattern that is very similar to a SSEH. This pattern is called a *d-trio* [Ung69].

Definition 6 *A flow table contains a* **d-trio** *for the transition from total state* (i, A) *to total state* (j, B) *if and only if there exists state* k *such that* $k = N(j, A)$, $k \neq i$, $k \neq j$, *and* $N(k, B) = j$.

A d-trio is not considered an essential hazard because a circuit that contains a d-trio, if designed properly and allowed to settle, will not reach an incorrect state. In [Ung69], Unger shows a procedure that produces a USTT state assignment that leads to a circuit which will reach the correct state even in the presence of SHs.

It's important to note that, if a d-trio manifests, there will be glitches in one or more state variables, thus increasing the time that the circuit needs to settle down. Also, there is no guarantee that the outputs will not glitch if a d-trio manifests. For these reasons, d-trios should be treated as hazards and be eliminated whenever possible.

Due to the similarity between d-trios and SSEHs, the techniques used in this paper to analyze SSEHs and to eliminate apparent and potential SSEHs work effectively with d-trios. No further mention of d-trios will be made.

3.5 Apparent Steady State Essential Hazards

Consider the transition from total state (2,010) to total state (3,011) in flow table ex1, shown in Table 1. This transition looks like a SSEH. The transition involves states 2, 3, and 4. As shown in the pair chart, states 3 and 4 are compatible and they can be merged. If, during state minimization, a minimal closed cover is selected such that one of the compatibles in the cover contains states 3 and 4, then the pattern will not be present in the reduced flow table. The three states have been reduced to two and they cannot constitute a SSEH.

This type of transition, which is present in *unminimized* flow tables only, is called an *apparent steady state essential hazard* (ASSEH). It resembles a SSEH but it involves at least two compatible states. If the proper cover is selected, the ASSEH is eliminated. If, on the other hand, an incorrect cover is selected (i.e. no compatible in the cover includes two states that contribute to the apparent hazard), then the ASSEH becomes a SSEH.

Definition 7 *An unminimized flow table contains an* **apparent steady state essential hazard** *for the transition from total state* (i, A) *to total state* (j, B) *if and only if there exists state* k *such that* $k = N(j, A)$, $k \neq i$, $k \neq j$, *and* $k \sim i$ *or* $k \sim j$.

Flow table ex1 contains another ASSEH for the transition from total state (5,010) to total state (3,011). It involves states 5, 3, and 4. This ASSEH can also be eliminated if any two of these states are merged. If compatible 345 is included in the solution, both apparent SSEHs are eliminated. The optimal state minimization is to merge states 2 and 6, and states 3, 4, and 5. The resulting reduced flow table contains only three states as shown in Table 3.

The following lemma states necessary and sufficient conditions to eliminate an ASSEH:

Lemma 7 *An ASSEH, constituted by states* i, j *and* k, *where* i *is the start state,* j *is the destination state, and* k *is the intermediate state, can be eliminated if and only if there exists a compatible* C *in the selected cover such that:*

1. $i \sim k$ *and* $i, k \in C$, *or*
2. $j \sim k$ *and* $j, k \in C$.

(i, j, k) is called a *required item* and we say that compatible C *properly covers* the required item. To eliminate all ASSEHs from a flow table, the selected cover must satisfy the following condition:

Definition 8 *A cover δ satisfies the* **required condition** *if and only if every required item is properly covered by a compatible in δ.*

3.6 Potential Steady State Essential Hazards

In an *incompletely* specified flow table, some patterns that include don't care entries can become SSEHs if the wrong value is assigned to one or more don't care next state entries. This patterns are called *potential steady state essential hazards.*

Definition 9 *A flow table contains a* **potential steady state essential hazard** *for the transition from total state (i, A) to total state (j, B) if and only if $N(j, A) = X$.*

The flow table shown in Figure 3(e) illustrates a PSSEH. Depending on the value given to the don't care next state entry in total state (j, A), a PSSEH can be eliminated or can become an ASSEH or a SSEH. If the don't care next state entry is specified as k such that $k \neq i$, $k \neq j$, $k \not\sim i$ and $k \not\sim j$ then the PSEEH becomes a SSEH. If the don't care next state entry is specified as k such that $k \neq i$, $k \neq j$, and $k \sim i$ or $k \sim j$ then the PSSEH becomes an ASSEH. On the other hand, If the don't care next state in total state (j, A) is specified as i or j, there is no risk of a SSEH, thus the PSSEH is eliminated.

If PSSEHs are not eliminated during state minimization, the reduced flow table can contain SSEHs that were not present in the initial specification. For example, the flow table shown in Table 3 corresponds to a minimal closed cover of ex1. However, it contains a steady state essential hazard for the transition from total state (a,000) to total state (b,001). This is caused by a PSSEH present in ex1 for the transition from (1,000) to (2,001). This PSSEH became a SSEH because, during state minimization, the don't care state in entry (2,000) was specified as state 3.

To avoid introducing this hazard, compatibles 26 and 345 cannot be both included in the solution, that is, they interfere with each other. We say that two compatibles **interfere** with each other if their simultaneous presence in a cover introduces one or more SSEHs in the reduced flow table.

The following lemma presents necessary and sufficient conditions to determine when two compatibles interfere with each other:

Lemma 8 *Two compatibles $C1$ and $C2$ interfere with each other if and only if they satisfy all of the following conditions:*

1. There exists a transition from total state (i, A) to total state (j, B) such that $i \notin C2$, $j \in C1$, $j \notin C2$ and $N(j, A) = X$,

2. There exists state r such that $r \in C1$, $r \neq i$, $r \neq j$, $N(r, A) = k$ and $k \in C2$.

To avoid introducing SSEHs during state minimization, the selected cover must not include compatibles that interfere with each other, that is, the selected cover must satisfy the following condition:

Definition 10 *A cover δ satisfies the* **interference-free condition** *if and only if no compatibles in δ interfere with each other.*

In the example above, state 2 is the key to the PSSEH because it contains the don't care next state entry. Compatible 26 is called the *major culprit* because it contains state 2.

An interference relation can be broken if the compatibles that interfere can be split. However, splitting compatibles may be a complicated and computationally intensive process[2]. An alternative is to redefine the prime compatibles.

Definition 11 *A compatible C_i is said to be* **EHF-prime** *if*
1. *C_i is prime.*
2. *C_i is non-prime and there exists C_j such that C_j is a major culprit and $C_i < C_j$.*

In example ex1, compatibles 2 and 6 are non-prime since $\Phi(26) = \emptyset$. However, compatibles 2 and 6 are EHF-prime because they are subsets of compatible 26, which is a major culprit. The EHF-prime compatibles are listed in the fifth column of Table 2.

4 Essential-hazard-free state minimization

4.1 Essential-hazard-free Cover

The following lemma follows from Lemma 3 and Lemma 6:

Lemma 9 *A flow table is essential-hazard-free (EHF) if and only if every transition in the flow table is TEH-free and SSEH-free.*

Now we can define an EHF solution for state minimization as follows:

Definition 12 *A cover δ is an* **EHF minimal closed cover** *if and only if it satisfies all of the following conditions:*

1. *Covering condition: δ covers every state of the flow table,*
2. *Closure condition: δ is closed,*
3. *Required condition: every required item is properly covered by a compatible in δ,*
4. *Interference-free condition: no compatibles in δ interfere with each other, and*
5. *Minimal condition: No other set of compatibles satisfies the above conditions and has fewer compatibles.*

The EHF state minimization problem can be stated as follows: Given an incompletely specified flow table and a set of SIC input transitions, find an EHF minimal closed cover.

[2] Which compatibles should be split and how to split them so that the solution is still optimal.

4.2 Elimination of potential Transient Essential Hazards

As described in the previous section, PTEHs are eliminated by constraining the output functions of the original specification. The algorithm which adds output constraints to prevent PTEHs from becoming TEHs during state minimization consists of two steps: first, identify all transition trios which contain PTEHs, and second, for each output function, set the don't care output to the proper value to prevent an EHs. The details of the algorithm are shown below:

Algorithm 1 *Constraining the original flow table:*

```
Input : A flow table and a list of input transitions.
Output: A constrained flow table.
Method:
  Build_Constrained_FT()
  {
     for each transition in the list of input transitions
        list all the transition trios
     for each transition trio, t={(i,A), (k,C), (j,B)}, in the
        list of transition trios
        for each output literal Zn
           if Zn(i,A) = Zn(j,B) and Zn(k,C) is a don't care
           then set Zn(k,C) = Zn(i,A)
  }
```

4.3 Constructing the Required and Interference Lists

We showed in the previous section that we can eliminate an ASSEH by merging any two states which contribute to the hazard. A required item contains the information needed to eliminate a ASSEH. The *required list* is a collection of all required items.

We also showed that some avoidable SSEHs may be introduced during state minimization due to the presence of PSSEHs. The information needed to avoid these SSEHs (i.e. the compatibles that interfere with each other) must be collected in a list called the *interference list*.

The algorithm which constructs the required and interference lists is shown below:

Algorithm 2 *Constructing the required and interference lists:*

```
Input : The Constrained flow table and the list of input transitions.
Output: The required and interference lists.
Method:
  Build_PLFL()
  {
      for each transition from (i,A) to (j,B), in the list of
          input transitions
          if i <> j then
          {
             if N(j,A) = k, k <> i and k <> j
             then add (i,j,k) to the required list

             if N(j,A) is a don't care then
             {
                for every compatible C1 in Prime Compatibles
```

```
                if j in C1 and exists r in C1 such that r <> j
                   N(r,A) = k, k in C2, and i and j not in C2
                then add (C1,C2) to the interference list
                /* Note that C1 is the major culprit */
             }
          }
    }
```

4.4 EHF-MinCover Algorithm

The EHF state minimization process is a modified version of state minimization. It is similar to the Puri method for efficiently searching for minimal closed covers [PG93]. The Puri method constructs a search tree from prime compatibles and builds up a tree-like search space by utilizing a tight lower bound derived from the maximal incompatibles. The tree is expanded if a solution is not found for the current lower bound.

Our algorithm consists of the following steps:

1. Construct the constrained flow table by applying Algorithm 1.

2. Generate the maximal incompatibles and prime compatibles.

3. Construct the *required* and *interference* lists by applying Algorithm 2.

4. Generate the EHF-prime compatibles.

5. Generate the cover table:

   For each compatible, C, in the set of EHF-prime compatibles, if state s in C, then insert C into entry s of the cover table, CT[s].

6. Set the lower bound to the number of states in the largest maximal incompatible and set the upper bound to the total number of states.

   Since the states in a maximal incompatible must be covered by different compatibles (i.e. a compatible cannot cover two incompatible states) and the compatibles in a minimal closed cover must cover all states, the lower bound of a minimal closed cover is equal to the number of states in the maximal incompatible with the maximum number of states.

   It is obvious that the upper bound is equal to the total number of states. An upper bound is used to evaluate the termination condition of state minimization: when the current lower bound is greater than the upper bound then the algorithm terminates. Note that there is always a solution for traditional state minimization (i.e. the unreduced flow table is a solution) but there may be no solution for EHF state minimization.

7. Based on the maximal incompatibles, generate a maximal incompatible search tree (MIST).

   A MIST tree is used to find a minimal solution. A path in the tree is a maximal incompatible. A node in the path is a state of the maximal incompatible. The MIST is constructed as follows: For each maximal incompatible generate a corresponding path such that for any two states i and j in a maximal incompatible, if $i < j$ then state i is a parent node of state j.

8. For each path in the maximal incompatible search tree, find all the possible candidates.
   (a) For each state, s, in the path, choose a compatible in CT[s] and put it into *candidate*.
   (b) put *candidate* into *candidates* and repeat the above operation to find next candidate.
9. For each candidate, check if the candidate satisfies the covering, closure, required, and interference-free conditions. If it does then the EHF minimal closed cover is found and returned.
10. increment the lower bound.
11. If lower bound $<=$ upper bound then expand the maximal incompatible search tree.
12. goto step 8.

The EHF-MinCover algorithm is shown in Algorithm 3 in appendix A.

4.5 An Example

Flow table ex1 is used to illustrate how the EHF state minimization algorithm works.

Step 1: Construct the constrained flow table: Since flow table ex1 is TEH-free, no output constraints are added.

Step 2: Generate the maximal incompatibles and prime compatibles: the maximal incompatibles are $\{123, 124, 125, 146, 156\}$ and the prime compatibles are $\{1, 26, 345, 36\}$.

Step 3: Construct the *required* and *interference* lists: the required list is $\{(2,4,3),(5,4,3)\}$ and the interference list is $\{(26,36),(26,345)\}$.

Step 4: Generate the EHF-prime compatibles: the EHF-prime compatibles are $\{1, 2, 26, 345, 36, 6\}$

Step 5: Generate the cover table: The cover table is shown in Table 6.

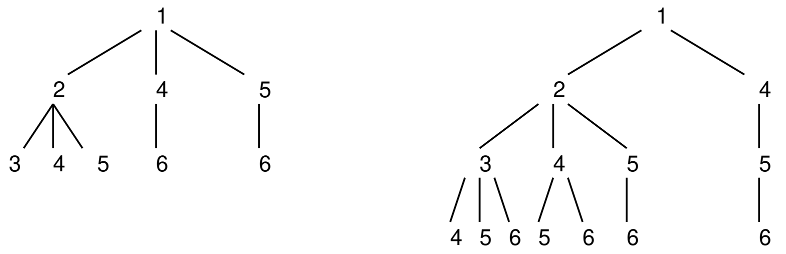
Cover Table	
state	EHF-Prime Compatibles
1	$<1>$
2	$<26><2>$
3	$<345><36>$
4	$<345>$
5	$<345>$
6	$<26><36><6>$

Table 6: The Cover Table of ex1.

Step 6: Set the lower and upper bounds: The lower bound is 3 and the upper bound is 6.

Step 7: Generate the maximal incompatible search tree (MIST) with bound = 3: The MIST with bound = 3 is shown in Figure 4(a).

Iteration steps(8-11): Try to find an EHF minimal closed cover with 3 EHF-prime compatibles.



(a) Search Tree with Lower Bound = 3 (b) Search Tree with Lower Bound = 4

Figure 4: Maximal Incompatible Search Trees:

For incompatible 123, the possible candidates are {1, 26, 345}, {1, 26, 36}, {1, 2, 345} and {1, 2, 36}. None of them is an EHF minimal close cover, for the first candidate violates the interference-free condition and the second through forth candidates violate the covering condition. It is not difficult to show that there is no 3-row EHF solution. Since there is no 3-row EHF solution, the MIST is expanded one more level. The MIST with bound = 4 is shown in Figure 4(b).

Iteration steps(8-11): Try to find an EHF minimal closed cover with 4 EHF-prime compatibles.

EHF-MinCover finds an EHF minimal closed cover, $\{1, 2, 345, 6\}$. The EHF minimal flow table is shown in Table 7.

	$x_1x_2x_3$ 000	001	011	101	010
a (1)	$\boxed{a}$,00	b,- -	-,- -	-,- -	$\boxed{a}$,10
b (2)	-,- -	$\boxed{b}$,11	c,- -	-,- -	$\boxed{b}$,11
c (345)	$\boxed{c}$,00	$\boxed{c}$,00	$\boxed{c}$,00	d,- -	$\boxed{c}$,00
d (6)	c,- -	$\boxed{d}$,11	-,- -	$\boxed{d}$,11	-,- -

Table 7: The Essential-Hazard-Free minimal Flow Table of ex1

It's important to note that there is a PSSEH in the transition from total state (a,000) to total state (b,001) in the reduced flow table for ex1. This PSSEH was prevented from becoming a SSEH during state minimization but was not eliminated. It can be eliminated by specifying the don't care next state entry in (b,000) as a or b. The choice of a or b should be decided during the state assignment or logic minimization stages.

5 Experimental Results

EHF-MinCover has been implemented in C++ and runs under Unix. Test cases were run on a Sun IPX workstation. These test cases are self-timed building blocks from [Ung93, Sut89, Bru91] and some of them are re-implemented using two-phase handshaking.

Table 8 shows the main results of the EHF state minimization. N_{in}, N_{out} and N_{IT} are the number of inputs, outputs and input transitions, respectively. S_{initSt} and S_{ReduSt} are the number of states before and after minimization, respectively. An NA in column N_{RedSt} means that no EHF solution was found by EHF-MinCover. N_{PC} and N_{MIC} are the numbers of prime compatibles and maximal incompatibles. N_{RL} and N_{IL} are the number of terms in the required and interference lists, respectively.

Four examples (ifelse, until, while, and two-step) are reduced to single state flow tables, and thus degenerate into combinational logic. Three examples (toggle, convert2-4, and convert4-2) have no EHF solution, i.e. the functions specifying these elements contain real essential hazards which can not be eliminated.

Example	N_{in}	N_{out}	N_{IT}	N_{InitSt}	N_{RedSt}	N_{PC}	N_{MIC}	N_{RL}	N_{IL}	EHF Sol
ex1	3	2	15	6	4	2	5	2	2	Yes
ex2	2	2	8	3	2	2	2	0	0	Yes
call	3	3	16	16	4	24	1	0	0	Yes
convert2-4	2	2	6	6	NA	3	8	2	0	No
convert4-2	2	2	6	6	NA	3	8	2	0	No
transitional-demux	3	2	36	16	4	81	1	16	0	Yes
ifelse	5	4	20	16	1	1	0	0	0	Yes
join	2	1	8	6	2	4	7	4	0	Yes
RSFF	2	2	16	8	4	4	16	8	0	Yes
storage-element	3	3	96	32	8	72	1420	32	0	Yes
toggle	1	2	4	4	NA	4	1	4	0	No
two-step	4	3	6	12	1	1	0	0	0	Yes
until	4	3	16	12	1	1	0	0	0	Yes
while	3	3	16	12	1	1	0	0	0	Yes

Table 8: Results of EHF MinCover.

Table 9 shows the results of state minimization by SIS [SSL+92, HRSJ91] and EHF-MinCover. The Unix *time* command is used to measure the running time of EHF-MinCover. T_{real} is the "wall-clock" time, T_{user} the time running in user-mode and T_{sys} the time running in system-mode in Unix. The experiments show that EHF-MinCover is very efficient. All cases take less than 1 minute to find EHF solutions or to report that no solution exists.

The time listed for SIS, under T_{CPU}, measures only the state minimization step, whereas the time reported for EHF-MinCover measures not only the time for EHF state minimization but also the time for reading the flow table, completing the hazard analysis, and converting the flow table to kiss format.

N_{SSEH} and N_{TEH} show the number of SSEHs and TEHs, respectively, in the reduced flow tables produced by SIS state minimization. Of the examples which do not degenerate into combinational logic, only two of the ASM flow tables (RSFF and storage-element) solved by SIS are EHF. In fact, in these two examples, the covers found by SIS and EHF-MinCover are exactly the same. For the rest of the examples, the solutions found by SIS contain either SSEHs or TEHs.

In some flow tables, the number of prime compatibles may be relatively large. In these cases, some heuristics (e.g. use maximal compatibles instead of prime compatibles) may be applied to avoid generating all the prime compatibles.

		EHF-MinCover				State Minimization of SIS			
Example	N_{InitSt}	N_{RedSt}	T_{real}	T_{user}	T_{sys}	N_{RedSt}	T_{CPU}	N_{SSEH}	N_{TEH}
ex1	6	4	0.55	0.10	0.16	3	0.00	1	0
ex2	3	2	0.56	0.05	0.16	2	0.00	0	1
call	12	4	1.41	0.45	0.13	2	0.03	0	4
convert2-4	6	NA	0.75	0.06	0.20	2	0.01	0	2
convert4-2	6	NA	0.63	0.13	0.13	2	0.02	0	2
demultiplexer	16	4	7.88	6.15	0.30	4	0.28	1	0
ifelse	16	1	0.85	0.26	0.15	1	0.01	0	0
join	6	2	0.65	0.06	0.15	2	0.01	0	0
RSFF	8	4	0.98	0.15	0.16	4	0.01	0	0
storage-element	32	8	45.76	41.08	2.05	8	0.24	0	0
toggle	4	NA	0.73	0.03	0.11	4	NA	4	0
two-step	12	1	0.63	0.05	0.13	1	0.00	0	0
until	12	1	0.73	0.16	0.13	1	0.01	0	0
while	12	1	0.98	0.16	0.13	1	0.00	0	0

Table 9: Comparison of State Minimization of SIS and EHF-MinCover.

6 Conclusions

This paper proposes an algorithm for essential-hazard-free state minimization of incompletely specified asynchronous sequential machines. Novel techniques to remove apparent and potential essential hazards are exploited in our algorithm. We also show that the existing state minimization methods introduce avoidable steady state as well as transient essential hazards during the state merging process.

This work is important because a normal flow table has no hazard-free realization under unbounded delay assumption if it contains any essential hazard. One promising result obtained is that most of the building block elements in [Ung93, Sut89, Bru91] can be reduced to EHF flow tables. To synthesize a hazard-free asynchronous circuits, a critical race free state assignment [Tra66] and hazard-free logic minimization [ND92] must be applied to the EHF reduced flow table generated by EHF-MinCover.

References

[Bru91] Erik Brunvand. *Translating Concurrent Communicating Programs into Asynchronous Circuits*. PhD thesis, Carnegie Mellon University, 1991.

[HRSJ91] G.D. Hachtel, J.K. Rho, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. In *Proc. European Design Automation Conf.*, pages 184–191. IEEE Computer Society Press, 1991.

[LKSV91] Luciano Lavagno, Kurt Keutzer, and Alberto Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proc. ACM/IEEE Design Automation Conf.*, pages 302–308, 1991.

[Mar86] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[MBM89] Teresa H.-Y. Meng, Robert W. Brodersen, and David G. Messerschmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Trans. on Computer-Aided Design*, 8(11):1185–1205, November 1989.

[ND91] Steven M. Nowick and David L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *Proc. Int'l. Conf. Computer-Aided Design*, pages 318–321. IEEE Computer Society Press, November 1991.

[ND92] Steven M. Nowick and David L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. In *Proc. Int'l. Conf. Computer-Aided Design*, pages 626–630. IEEE Computer Society Press, November 1992.

[PG93] Ruchir Puri and Jun Gu. An efficient algorithm to search for minimal closed covers in sequential machines. *IEEE Trans. on Computer-Aided Design*, 12(6):737–745, June 1993.

[SSL+92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoy, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[Sut89] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, January 1989.

[Tra66] James H. Tracey. Internal state assignment for asynchronous sequential circuits. *IEEE Trans. Electronic Computers*, EC-15(4):551–560, August 1966.

[Ung68] Stephen H. Unger. A row assignment for delay-free realizations of flow tables without essential hazards. *IEEE Transactions on Computers*, 17(2):146–151, February 1968.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[Ung93] S. H. Unger. A building block approach to unclocked systems. In *Proc. Hawaii International Conf. System Sciences*, pages 339–348. IEEE Computer Society Press, January 1993.

Appendix A:

Algorithm 3 *EHF-MinCover:*

```
Input : A flow table and a list of input transitions.
Output: An EHF minimal closed cover.
Method:
   EHFMinCover()
   {
      /* Construct a constrained flow table by applying Algorithm 1. */
      Build_Constrained_FT();
      Generate MAXimal InCompatibles (MAXIC) and Prime Compatibles (PC);

      /* Construct the required and forbidden lists by applying Algorithm 2. */
      Build_PLFL();
      Generate EHF-Prime Compatibles (EHF-PC);

      /* Generate a Cover Table, CT, from the EHF-Prime Compatibles */
      for each compatible C in EHF-PC
         for each state s
            if s in C then add C to CT[s];

      set the LowerBound and UpperBound;
      generate Maximal Incompatible search tree(MIST);
      While (TRUE) {
         for each path MI in MIST {
            while(select_Candidate(MI)) {
               if Candidate satisfies all of the following conditions:
                  1. Covering condition: Candidate covers all states of the flow table;
                  2. Closure  condition: Candidate is closed;
                  3. Required condition: Every required item is properly covered
                                         by a compatible in Candidate;
                  4. Interference-free condition: No compatibles in Candidate
                                                  interfere with each other;
               then {
                  solution = shrink the candidate;
                  return candidate;
               } /* end of if check solution           */
            }    /* end of while there is an candidate */
         }       /* end of for each MI                 */

         LowerBound = LowerBound +1;
         if (LowerBound > UpperBound)
            No EHF Solution is found and exit;
         else
            Expand MIST with LowerBound;
      } /* end of while */
   }
```

```
/* select_Candidate() function enumerates all the possible candidates from
 * CT such that each candidate contains all the states of MI.
 */
select_Candidate(MI)
{
   /* A candidate is a set of compatibles */
   for each state s in MI {
      select a compatible from CT[s];
      put the compatible into candidate;
   }
   if no new Candidate can be found
   then return FALSE;
   else return TRUE;
}
```