Espresso-HF: A Heuristic Hazard-Free Minimizer for Two-Level Logic*

Michael Theobald Steven M. Nowick Tao Wu

Department of Computer Science
Columbia University
New York, NY 10027

Abstract — We present a new heuristic algorithm for hazard-free minimization of two-level logic. On nearly all examples, the algorithm finds an exactly minimum-cost cover. It also solves several problems which have not been previously solved using existing exact minimizers. We believe this is the first heuristic method based on Espresso to solve the general hazard-free two-level minimization problem, for multiple-input change transitions.

1 Introduction

Asynchronous design has been the focus of much recent research activity. A number of methods have been developed for the design of hazard-free controllers [5, 2, 6, 13]. These methods have been applied to several large and realistic design examples, including a low-power infrared communications chip [3], a second-level cache-controller [7], and an implementation of a SCSI controller [12].

An important aspect of these methods is the development of optimized CAD tools. In synchronous design, the development and implementation of CAD packages has been critical to the success of modern automated digital design. In asynchronous design, much progress has been made, including tools for: exact hazard-free two-level logic minimization [9], optimal state assignment [2] and synthesis-for-testability [8]. However, these tools have been limited in handling large-scale designs. In particular, while the exact hazard-free minimization algorithm has been effective on small- and medium-sized examples, it has been unable to produce solutions for several large design problems [2].

In this paper, we present an algorithm to solve the heuristic hazard-free two-level logic minimization problem. The method is heuristic solely in terms of the cardinality of solution. In all cases, it guarantees a hazard-free solution. It also implements both single-output and multi-output minimization. The algorithm is based on Espresso [10, 1], but with a number of significant modifications to handle hazard-free constraints. The algorithm, called Espresso-HF, also employs a new and much more efficient algorithm to check for existence of a solution, without generating all prime implicants.

Our prototype can solve all examples that we have available so far, and almost always obtains an exactly minimum cover. It also solves several examples which have not been solved before.

The paper is organized as follows. Section 2 gives background on circuit models, hazards and hazard-free minimization. Section 3 describes the Espresso-HF algorithm. Section 4 describes an algorithm to determine if a hazard-free solution exists. Section 5 presents experimental results, and Section 6 gives conclusions.

*This work was supported by NSF under Grant no. MIP-9308810 and by an Alfred P. Sloan Research Fellowship.

2 Background

The material of this section focuses on hazards and hazard-free logic minimization, and is taken from [2] and [9]. For simplicity, we focus on single-output functions. A generalization of these definitions to multi-output functions is straightforward, and is described in [2]. Our heuristic minimizer handles multi-output functions.

2.1 Circuit Model

This paper considers combinational circuits having arbitrary finite gate and wire delays (*unbounded wire delay model* [9]). A pure delay model is assumed as well (see [11]).

2.2 Multiple-Input Changes

Definition 2.1 *Let A and B be two minterms. The* **transition cube**, *$[A, B]$, from A to B has* **start point** *A and* **end point** *B, and contains all minterms that can be reached during a transition from A to B. More formally, if A and B are described by products, with i-th literals A_i and B_i, respectively, then the i-th literal for the product of $t = [A, B]$ is the Boolean function $A_i + B_i$. An* **input transition** *or* **multiple-input change** *from input state (minterm) A to B is described by transition cube $[A, B]$.*

A multiple-input change specifies what variables change value and what the corresponding *starting* and *ending* values are. Input variables are assumed to change simultaneously. (Equivalently, since inputs may be skewed arbitrarily by wire delays, inputs can be assumed to change monotonically in any order and at any time.) Once a multiple-input change occurs, no further input changes may occur until the circuit has stabilized. In this paper, we consider only transitions where f is fully defined; that is, for every $X \in [A, B]$, $f(X) \in \{0, 1\}$.

2.3 Function Hazards

A function f which does not change monotonically during an input transition is said to have a *function hazard* in the transition.

Definition 2.2 *A function f contains a* **static function hazard** *for the input transition from A to C if and only if: (1) $f(A) = f(C)$, and (2) there exists some input state $B \in [A, C]$ such that $f(A) \neq f(B)$.*

Definition 2.3 *A function f contains a* **dynamic function hazard** *for the input transition from A to D if and only if: (1) $f(A) \neq f(D)$; and (2) there exist a pair of input states, B and C, such that (a) $B \in [A, D]$ and $C \in [B, D]$, and (b)$f(B) = f(D)$ and $f(A) = f(C)$.*

If a transition has a function hazard, no implementation of the function is guaranteed to avoid a glitch during the transition, assuming arbitrary gate and wire delays [9, 11]. Therefore, we consider only transitions which are free of function hazards.

2.4 Logic Hazards

If f is free of function hazards for a transition from input A to B, an implementation may still have hazards due to possible delays in the logic realization.

Definition 2.4 *A circuit implementing function f contains a* **static (dynamic) logic hazard** *for the input transition from minterm A to minterm B if and only if: (1) $f(A) = f(B)$ ($f(A) \neq f(B)$), and (2) for some assignment of delays, the circuit's output is not monotonic during the transition interval.*

33rd Design Automation Conference ®

Permission to make digital/hard copy of all or part of this work for personal or class-room use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permssion and/or a fee.
DAC 96 - 06/96 Las Vegas, NV, USA ©1996 ACM, Inc. 0-89791-833-9/96/0006..$3.50

That is, a static logic hazard occurs if $f(A) = f(B) = 1$ (0), but the circuit's output makes an unexpected $1 \rightarrow 0 \rightarrow 1$ ($0 \rightarrow 1 \rightarrow 0$) transition. A dynamic logic hazard occurs if $f(A) = 1$ and $f(B) = 0$ ($f(A) = 0$ and $f(B) = 1$), but the circuit's output makes an unexpected $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ ($0 \rightarrow 1 \rightarrow 0 \rightarrow 1$) transition.

2.5 Conditions for a Hazard-Free Transition

We now describe conditions to ensure that a sum-of-products implementation, F, is hazard-free for a given input transition [9]. Assume that $[A, B]$ is the transition cube corresponding to a *function-hazard-free* transition from input state A to B for a function f.

Lemma 2.5 *If f has a $0 \rightarrow 0$ transition in cube $[A, B]$, then the implementation is free of logic hazards for the input change from A to B.*

Lemma 2.6 *If f has a $1 \rightarrow 1$ transition in cube $[A, B]$, then the implementation is free of logic hazards for the input change from A to B* if and only if *$[A, B]$ is contained in some cube of cover F (i.e.,* some *product must hold its value at 1 throughout the transition).*

The conditions for the $0 \rightarrow 1$ and $1 \rightarrow 0$ cases are symmetric. Without loss of generality, we consider only a $1 \rightarrow 0$ transition, where f(A)=1 and f(B)=0.[1]

Lemma 2.7 *If f has a $1 \rightarrow 0$ transition in cube $[A, B]$, then the implementation is free of logic hazards for the input change from A to B* if and only if *every cube $c \in F$ intersecting $[A, B]$ also contains A (i.e., no product may glitch* in the middle *of a $1 \rightarrow 0$ transition).*

Lemma 2.8 *If f has a $1 \rightarrow 0$ transition from input state A to B which is hazard-free in the implementation, then, for every input state $X \in [A, B]$ where $f(X) = 1$, the transition subcube $[A, X]$ is contained in some cube of cover F (i.e., every $1 \rightarrow 1$ sub-transition must be free of logic hazards).*

2.6 Required and Privileged Cubes

The cube $[A, B]$ in Lemma 2.6 and the *maximal* subcubes $[A, X]$ in Lemma 2.8 are called *required cubes*. Each required cube *must* be contained in some cube of cover F to ensure a hazard-free implementation. More formally:

Definition 2.9 *Given a function f, and a set, T, of specified function-hazard-free input transitions of f, every cube $[A, B] \in T$ corresponding to a $1 \rightarrow 1$ transition, and every maximal subcube $[A, X] \subset [A, B]$ where f is 1 and $[A, B] \in T$ is a $1 \rightarrow 0$ transition, is called a* **required cube**.

Lemma 2.7 constrains the cubes which may be included in a cover F. Each $1 \rightarrow 0$ transition cube is called a *privileged cube*, since no cube c in the cover may intersect it unless c contains its *start point.* If a cube intersects a privileged cube but does not contain its start point, it *illegally intersects* the privileged cube and may not be included in the cover. More formally:

Definition 2.10 *Given a function f, and a set, T, of specified function-hazard-free input transitions of f, every cube $[A, B] \in T$ corresponding to a $1 \rightarrow 0$ transition is called a* **privileged cube**.

2.7 Hazard-Free Covers

A *hazard-free cover* of function f is a cover of f whose AND-OR implementation is hazard-free for a *given set*, T, of specified input transitions. (It is assumed below that the function is defined for all specified transitions; the function is undefined for all other input states.)

Theorem 2.11 (Hazard-Free Covering) *A sum-of-products F is a hazard-free cover for function f for the set T of specified input transitions if and only if:*

(a.) No cube of F intersects the OFF-set of f;

(b.) Each required cube *of f is contained in some cube of F; and*

(c.) No cube of F intersects any privileged cube *illegally.*

[1] A $0 \rightarrow 1$ transition from A to B has the same hazards as a $1 \rightarrow 0$ transition from B to A.

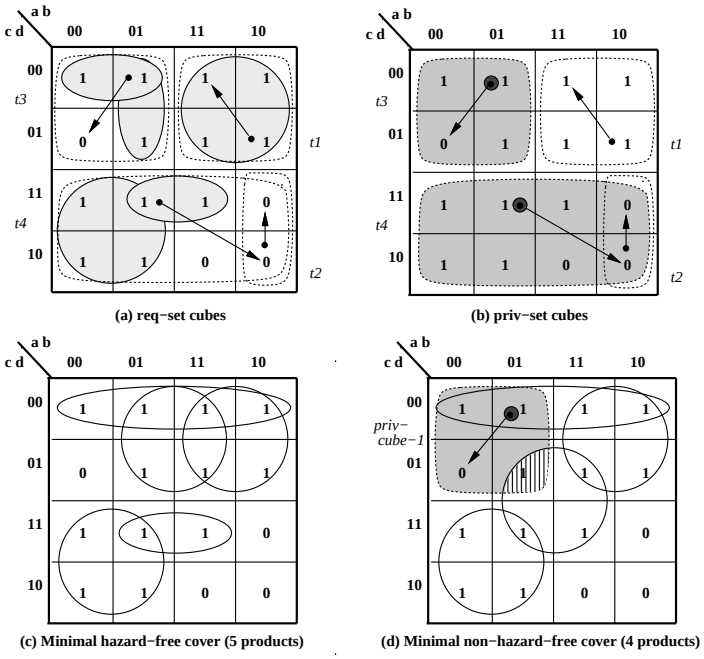

Figure 1: Hazard-Free Minimization Example

Theorem 2.11(a) and (c) determine the implicants which may appear in a hazard-free cover of a function f, called *dynamic-hazard-free (dhf-) implicants*.

Definition 2.12 *A* **dhf-implicant** *is an implicant which does not intersect any privileged cube of f illegally. A* **dhf-prime implicant** *is a dhf-implicant contained in no other dhf-implicant. An* **essential dhf-prime implicant** *is a dhf-prime implicant which contains a required cube contained in no other dhf-prime implicant.*

Theorem 2.11(b) determines the covering requirement for a hazard-free cover of f: *every required cube of f must be covered*, that is, contained in some cube of the cover. Thus, the **two-level hazard-free logic minimization problem** is *to find a minimum cost cover of a function using only dhf-prime implicants where every required cube is covered.*

In general, the covering conditions of Theorem 2.11 may not be satisfiable for an arbitrary Boolean function and set of transitions [11, 9]. This case occurs if conditions (b) and (c) cannot be satisfied simultaneously.

A hazard-free minimization example is shown in Figure 1.

2.8 Exact Hazard-Free Minimization Algorithm

A single-output exact hazard-free minimizer has been developed by Nowick and Dill [9]. It has recently been extended to hazard-free multi-valued minimization by Fuhrer, Lin and Nowick [2]. The latter method uses Espresso to generate all prime implicants, then transforms them into dhf-prime implicants, and finally employs Espresso's MINCOV to solve the resulting covering problem.

3 Heuristic Hazard-Free Minimization

3.1 Overview

The goal of heuristic hazard-free minimization is to find an optimal (but not necessarily exactly minimum) solution to the hazard-free covering problem. The basic minimization strategy of Espresso-HF for hazard-free minimization is similar to the one used by Espresso-II. However, we use additional constraints to ensure that the resulting cover is hazard-free, and the algorithms have significant differences.

One important distinction is in the use of unate recursive paradigm in Espresso-II, i.e. to decompose operations recursively leading to efficiently solvable sub-operations on unate functions.

```
Espresso-HF(f,T)
  Q = generate_set_of_required-cubes(f,T)
  P = generate_set_of_privileged-cubes(f,T)
  S = generate_set_of_start-points(f,T)
  R = OFF-set(f)
  Q^f = {supercube_dhf(q) | q ∈ Q}
  If "undefined" ∈ Q^f then no solution is possible; exit
  Minimize Q^f with respect to single cube containment
  F = Q^f
  (F, E) = expand_and_compute_essentials(F)
  Remove all cubes from Q^f that are already covered by E
  F = F - E
  F = irredundant(F)
  do
     φ2 = |F|
     do
           φ1 = |F|
           F = reduce(F)
           F = expand(F)
           F = irredundant(F)
     while (|F| < φ1)
     F = last_gasp(F)
  while (|F| < φ2)
  F = F ∪ E
  F = make_dhf_prime(F)
```

Figure 2: The Espresso-HF algorithm.

To the best knowledge of the authors, this paradigm cannot be applied directly to hazard-free minimization. We therefore follow the basic steps of Espresso-II, modified to incorporate hazard-free constraints, but without the use of unate recursive algorithms. Because of the constraints and granularity of the hazard-free minimization problem, high-quality results are still obtained even for large examples.

In this section, we describe the basic steps of the algorithm, concentrating on the new constraints that must be incorporated to guarantee a cover to be hazard-free. As in Espresso, the size of the cover is never increased in size. In addition, after an initial phase, the cover always represents a valid solution, i.e. a cover of f that is also hazard-free. Pseudocode is shown in Figure 2.

The first step of Espresso-HF is to read in PLA files specifying a Boolean function, f, and a set of specified function-hazard-free transitions, T. These inputs are used to generate the set of required cubes Q, the set of privileged cubes P and their start points S, and the off-set R. Generation of these sets is immediate from the earlier lemmas (see also [9]).[2]

Unlike Espresso, the given initial specification Q does *not* in general represent a solution: while Q is a cover of f, it is not necessarily hazard-free. Therefore, processing begins by expanding each required cube into the *uniquely defined* minimum dhf-implicant covering it.

The next step is to identify essential dhf-implicants using a modified EXPAND step. This algorithm uses a novel approach to identifying equivalence classes of implicants, each of which is treated as a single implicant. Essential implicants, as well as all required cubes covered by them, are then removed from F and Q^f, respectively, resulting in a smaller problem to be solved by the main loop. Before the main loop, the current cover is also made irredundant.

Next, as in Espresso, Espresso-HF applies the three operators REDUCE, EXPAND, and IRREDUNDANT to the current cover until no further improvement in the size of the cover is possible. Since the result may be a local minimum, the operator LAST_GASP is then applied to find a better solution using a different method. EXPAND uses new hazard-free notions of *essential parts* and *feasible expansion*. The other steps differ from Espresso as well.

At the end, there is an additional step to make the resulting implicants *dhf-prime*, since it is desirable to obtain a cover that consists of dhf-prime implicants. The motivation for this step will be made clear in the sequel.

[2]The algorithm does not need an explicit cover for the don't-care set because the operations use the off-set to check if a cube is valid.

3.2 Dhf-Canonicalization of Initial Cover

In Espresso, the initial cover of a function is provided by its ON-set, F^{ON}. This cover is a seed solution, which is iteratively improved by the algorithm. By analogy, in Espresso-HF, the initial cover is provided by the set of required cubes, Q. However, *unlike* Espresso, our initial specification does not in general represent a solution: though Q is a cover, it is not necessarily hazard-free. Therefore, processing begins by expanding each required cube into the *uniquely defined* minimum dhf-implicant containing it. This expansion represents a *canonicalization step*, transforming a potentially hazardous initial cover Q into a hazard-free initial cover Q^f.

Example. Consider the function f in the Karnaugh map of Figure 3. A set T of specified multiple-input transitions is indicated by arrows. There are two $1 \rightarrow 0$ transitions, each corresponding to a privileged cube: $p1 = a'c'$ (start point $p1_{strt} = a'bc'd'$) and $p2 = ad$ (start point $p2_{strt} = abc'd$). The initial cover is given by the set Q of required cubes: $\{a'c'd'$, $a'bc'$, ac', $ac'd$, abd, bcd, $bcd'\}$. This cover is hazardous. In particular, consider the required cube $r = bcd$, corresponding to the $1 \rightarrow 1$ transition from $abcd = 0111$ to 1111. Required cube r illegally intersects privileged cube $p2$, since it intersects $p2$ but does not contain $p2_{strt}$. To avoid illegal intersection, r must be expanded to the smallest cube which also contains $p2_{strt}$: $r^{(1)} = \text{supercube}(\{r, p2_{start}\})$. However, this new cube $r^{(1)} = bd$ now illegally intersects privileged cube $p1$, since it does not contain $p1_{strt}$. Therefore, cube $r^{(1)}$ in turn must be expanded to the smallest cube containing $p1_{strt}$: $r^{(2)} = \text{supercube}(\{r^{(1)}, p1_{start}\})$. The resulting expanded cube, $r^{(2)} = b$, has no illegal intersections and is therefore a dhf-implicant. □

In this example, $r^{(2)}$ is a hazard-free expansion of r, called a **canonical required cube**; it can therefore replace r in the initial cover. (Note that such a canonicalization is feasible if and only if the hazard-free covering problem has a solution; see Section 4.)

Thus, an initial set Q of required cubes is replaced by a set Q^f of canonical required cubes (after having been minimized with respect to single cube containment). Q^f is a valid hazard-free cover of the function to be minimized, and is used as an initial cover for the minimization process. In fact, Q^f has a second role as well: it is used to simplify the covering problem. In particular, Q^f defines a new covering problem: each cube of Q^f (*not* Q) must be contained in some dhf-implicant. It is straightforward to show that the two covering problems are equivalent: if a dhf-implicant p contains a required cube r in Q, p must also contain the canonical required cube of r in Q^f; if not, p would not be a dhf-implicant.

In the above example, any dhf-implicant which contains required cube $r = bcd$ must also contain canonical required cube $r^{(2)} = b$. Therefore, the hazard-free minimization problem is unchanged, but canonical required cubes are used. An advantage of using Q^f is that it may have smaller size than Q, i.e. being a more efficient representation of the problem. Also, since the cubes in Q^f are in general larger than the corresponding ones in Q, the EXPAND operation may be sped up.

In sum, the set of canonical required cubes Q^f replaces the set of required cubes Q as both (i) the initial cover, and (ii) the set of objects to be covered. Henceforth, the term "set of required cubes" will be used to refer to set Q^f.

We formalize the notion of canonicalization below.

Definition 3.1 *Let f be a Boolean function, T be a set of function hazard-free transitions, and C be a set of implicants. The* **dhf-supercube of** *C with respect to function f and transitions T, indicated as $supercube_{dhf}^{(f,T)}(C)$, is the smallest dhf-implicant containing the cubes of C.*

The superscript (f, T) is omitted when it is clear from the context. $supercube_{dhf}(C)$ is computed by the simple algorithm shown in

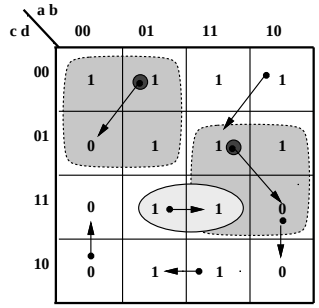

Figure 3: Canonicalization Example

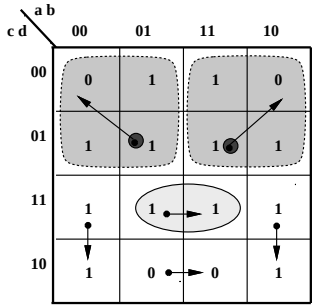

Figure 4: Essential Example

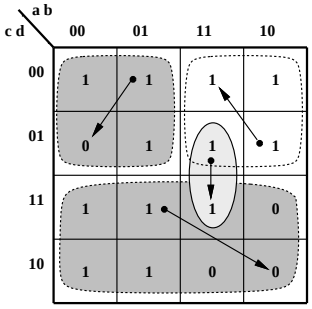

Figure 5: Existence Example

$supercube_{dhf}$ (set of cubes $C = \{c_1, \ldots, c_n\}$)
  $r = supercube(\{c_1, \ldots, c_n\})$
  while (r intersects any privileged cube p_i illegally)
    $r = supercube(\{r, s_i\})$ where s_i is the start point of p_i
  if r intersects the off-set then return "undefined" else return r

Figure 6: $Supercube_{dhf}$ computation

Expand_cube(cube a, req-set Q^f, priv-set P, cover-set F, off-set R)
  $F_a = F - a$
  $Q_a = Q^f$
  $P_a = P$
  $R_a = R$
  $free\_entries$ =complement_pos_cube_notation(a)
  while ($F_a \neq \emptyset$)
    update($a, free\_entries, F_a, Q_a, P_a, R_a$)
    $F_a = \{c \in F_a | supercube_{dhf}(\{a, c\})$ is defined $\}$
    Let c_b be the best candidate in F_a
    $a = supercube_{dhf}(\{a, c_b\})$
  while ($Q_a \neq \emptyset$)
    update($a, free\_entries, F_a, Q_a, P_a, R_a$)
    $Q_a = \{c \in Q_a | supercube_{dhf}(\{a, c\})$ is defined $\}$
    Let c_b be the best candidate in Q_a
    $a = supercube_{dhf}(\{a, c_b\})$

Figure 7: Expand (for a cube a)

Figure 6.

The *canonical required cube* of a required cube r can now be defined as the *dhf-supercube* of the set $C = \{r\}$. The computation of dhf-supercubes for larger sets will be needed to implement some of the operators presented in the sequel.

3.3 Expand

In Espresso, the goal of EXPAND is to enlarge each implicant in turn into a prime implicant. As an implicant is expanded, it may contain other implicants which can be removed, hence the cover cardinality is reduced. If the current implicant cannot be expanded to contain another implicant completely, then, as a secondary goal, the implicant is expanded to overlap as many other implicants of the current cover as possible.

In Espresso-HF, the primary goal is similar: to expand a dhf-implicant to contain as many other dhf-implicants as possible. However, EXPAND in Espresso-HF has two major differences. Unlike Espresso, expansion in some literal (*i.e.*, "raising of entries") may *imply* that other expansions be performed. That is, raising of entries is a *binate problem*, not a unate problem. Furthermore, Espresso-HF's EXPAND uses a different strategy for its secondary goal. By the Hazard-Free Covering Theorem, each required cube needs to be contained in some cube of the cover. Therefore, as a secondary goal, an implicant is expanded to contain as many required cubes as possible.

We now describe the implementation of EXPAND in Espresso-HF. Pseudocode for the expansion of a single cube is shown in Figure 7.

3.3.1 Determination of Essential Parts and Update of Local Sets

As in Espresso, a *free list* is maintained, to accelerate the expansion [10]. The free entries consist of all entries of the current implicant, in positional cube notation [4], which are still candidates to be raised to 1. Initially, a free entry is assigned a 1 (0) if the current implicant a has a 0 (1) in the corresponding position. An *overexpanded cube* is defined as the cube a where all free entries have been raised simultaneously.

An *essential part* is one which can never, or always, be raised[10]. Our definition of "essential parts" is different from Espresso, since a hazard-free cover must be maintained.

First, determine which entries can *never be raised* and remove them from $free$. This is achieved by searching for any cube in the off-set R that has distance 1 from a, using the same approach as in Espresso.

Next, determine which parts can *always be raised*, raise them and remove them from $free$. This step differs from Espresso. In Espresso, a part can always be raised if it is 0 in all cubes of the OFF-set, R. That is, it is guaranteed that the expanded cube will never intersect the OFF-set. In contrast, in Espresso-HF, we must ensure that an implicant is also *hazard-free*: it cannot intersect the OFF-set, nor can it illegally intersect a privileged cube. Unlike in Espresso, this is achieved by searching for any column that has only 0s in R AND where each 1 in P implies that the corresponding start point is covered by a.

Example. Figure 1(a) indicates the set of required cubes, which forms an initial hazard-free cover. Consider the cube bcd (11010101, in positional cube notation). As in Espresso, the 0-entries for literals b' and d' can never be raised, since the cube would intersect the OFF-set. However, after updating the free list, Espresso indicates that literal c' can *always* be raised, since the resulting cube will never intersect the OFF-set. In contrast, in Espresso-HF, raising c' results in an illegal intersection with privileged cube $a'c'$, so it cannot "always be raised". □

Since the hazard-free minimization is somewhat more constrained, the expansion of a cube a can be accelerated by the following new operations on the local sets: P_a, R_a, Q_a. These sets are associated with cube a, and are updated as expansion proceeds. (1) Remove privileged cubes from P where the corresponding start point is already covered by a (since no further checking is required for illegal intersection). (2) Move privileged cubes to the local off-set R_a if the overexpanded cube does not include the corresponding start points (since a can never be expanded to include these start points, and therefore must avoid intersection with the cubes entirely). (3) Move privileged cubes to the local off-set where $supercube_{dhf}(\{$a, start point$\})$ intersects the off-set (a can never be expanded to include these start points, and therefore must avoid intersection with the cubes entirely).

3.3.2 Detection of Feasibly Covered Cubes of F

In Espresso, a cube d in F is *feasibly covered* by a if supercube($\{a,d\}$) (the smallest cube containing both a and d) is an implicant. In Espresso-HF, this definition needs to be modified to insure *hazard-free covering*.

Definition 3.2 *A cube d in F is* **dhf-feasibly covered** *by a if $supercube_{dhf}(\{a,d\})$ is defined.*

This definition insures that the resulting cube is (i) an implicant (does not intersect OFF-set), and (ii) is also a dhf-implicant (does not intersect any privileged cube illegally). This definition canonicalizes the resulting supercube to produce a dhf-implicant. That is, $supercube_{dhf}(\{a,d\})$ may properly contain supercube($\{a,d\}$), since the former may be expanded through a series of *implications* in order to reach the minimum dhf-implicant which contains both a and d. Using this definition, the following is an algorithm to find dhf-feasibly covered cubes of F.

While there are cubes in F that are dhf-feasibly covered, iterate the following:

Replace a by $supercube_{dhf}(\{a, d\})$, where d is a dhf-feasibly covered cube such that the resulting cube will cover as many cubes of the cover as possible. Covered cubes are then removed, reducing the cover cardinality. Determine essential parts and update local sets (see above).

3.3.3 Detection of Feasibly Covered Cubes of Q^f

We continue to expand cube a even if it cannot cover any more cubes of F. This is motivated by the Hazard-Free Covering Theorem, that states that each required cube needs to be contained in some cube of the cover. Therefore, as a secondary goal, a cube a is expanded to contain as many required cubes as possible. The strategy used in this sub-step is similar to the one used in the preceding one, i.e. while there are cubes in Q^f that are dhf-feasibly covered, iterate the following:

Replace a by $supercube_{dhf}(\{a, r\})$, where r is a dhf-feasibly covered *required cube* such that the resulting cube will cover as many required cubes not already contained in a as possible. Covered required cubes are then removed. Determine essential parts and update local sets (see above).

3.3.4 Constraints on Hazard-Free Expansion

In Espresso, an implicant is expanded until no further expansion is possible, i.e. when the implicant is prime. Two steps are used: (i) expansion to overlap a maximum number of cubes still covered by the overexpanded cube; and (ii) raising of entries to find the largest prime implicant covering the cube.

In Espresso-HF, however, we do not implement these remaining EXPAND steps, based on the following observation. The result of our EXPAND steps guarantees that a dhf-implicant can never be further expanded to contain additional required cubes. Therefore, by the Hazard-Free Covering Theorem, no additional objects (required cubes) can be covered through further expansion. In contrast, in Espresso, further expansion steps *may* result in covering additional ON-set minterms. Because of this distinction, the benefits of further expansion are mitigated. Therefore, in general, our algorithm does not transform dhf-implicants into dhf-prime implicants. Since expansion to dhf-primes is important for literal reduction and testability, it is included as a final post-processing step: *make-dhf-prime* (see Figure 2).

3.4 Essentials

Essential prime implicants are prime implicants that need to be included in any cover of prime implicants. Therefore, it is desirable to identify them as soon as possible making the resulting problem size smaller. On the one hand, we know of no efficient solution for identifying the essential dhf-primes using the unate recursion paradigm as in Espresso. On the other hand, the hazard-free minimization problem is constrained by the notion of covering of *required cubes*, allowing a powerful method to classify essentials as equivalence classes.

Example. Consider Figure 4. The required cube, $r = bcd$, is covered by precisely two dhf-prime implicants: $p1 = bd$ and $p2 = cd$. Neither $p1$ nor $p2$ is an essential dhf-prime, since r is covered by both. And yet, clearly, either $p1$ or $p2$ (not both) must be included in any cover of dhf-primes. If we assume the standard cost function of cover cardinality, $p1$ and $p2$ are of equal cost. □

Our EXPAND method supports the notion of *equivalence classes*, since implicants are not expanded beyond the required cubes which they cover. In the above example, r would not be expanded further, since no feasible required cubes can be found. Cube r therefore represents an *essential equivalence class* corresponding to the set $\{bd, cd\}$ of dhf-primes. It should be removed from the cover. Using this strategy, which is applicable since the number of required cubes is usually not large, the problem size can often be reduced dramatically (see Section 5).

Espresso computes essentials after an initial expand and irredundant. In contrast, Espresso-HF computes essentials as part of a modified expand step. The algorithm is outlined as follows:

The algorithm starts with the initial hazard-free cover, Q^f, of required cubes. One seed cube is selected and expanded greedily using EXPAND, to a dhf-implicant p. This implicant is characterized by the set, Q^p, of required cubes which it contains. Dhf-implicant p is called an **essential equivalence class** if it contains some required cube, q^f, which cannot be expanded to any other equivalence class. To check if q^f can be expanded to a different equivalence class, a simple pairwise check is used: for each required cube s^f not covered by p, determine if $supercube_{dhf}(\{q^f, s^f\})$ is feasible for some s^f. If no feasible expansion exists for q^f, it is called a **distinguished required cube**, and therefore p is essential. Otherwise, the process is repeated for every required cube q^f in Q^p. When an essential p is identified, all required cubes covered by p are removed, and the covering problem is updated. This step can result in "secondary essential" equivalence classes. The procedure iterates until all essentials are identified.

3.5 Reduce

The goal of the REDUCE operator is to set up a cover that is likely to be made smaller by the following EXPAND step. To achieve this, each cube c in a cover F is maximally reduced in turn to a cube $\tilde{c}$, such that the resulting set of cubes, $\{F - c\} \cup \tilde{c}$ is still a cover.

Espresso uses the unate recursive paradigm to maximally reduce each cube. Since Espresso-HF is a required cube covering algorithm, there is no obvious way to use this paradigm. Fortunately, the hazard-free problem is more constrained, making it possible to use an efficient enumerative approach based on required cubes.

Our REDUCE algorithm is as follows. For each required cube $q^f \in Q^f$, it computes the number of cubes in the current cover F that contain q^f. The algorithm then reduces each cube c in the cover in order. A reduced cube is defined by the set of required cubes which c uniquely covers. Using the $supercube_{dhf}$ operator, this set is transformed into the maximally-reduced dhf-implicant $\tilde{c}$. We then update the number of cubes in the current cover F that cover each q^f, and similarly reduce the remaining cubes in F, in order.

3.6 Irredundant

Espresso uses the unate recursive paradigm to find an irredundant cover. However, there is no obvious way to employ this paradigm, since a "redundant cover" (according to covering of minterms) may in fact be irredundant with respect to covering of required cubes.

Therefore, as in REDUCE, our approach is required-cube based. Considering the Hazard-Free Covering Theorem, it is straightforward that IRREDUNDANT can be reduced to a covering problem of the cubes in Q^f by the cubes in F. That is, the problem reduces to a minimum-covering problem of (i) required cubes, using (ii) dhf-implicants in the current cover. In practice, the number of required cubes and cover cubes usually make the covering problem manageable. Espresso's MINCOV can be used to solve this covering problem exactly, or heuristically (using its heuristic option).

3.7 Last Gasp

The inner loop may lead to a suboptimal local minimum. LAST GASP then uses a different approach attempting to reduce the cover size. In Espresso, each cube $c \in F$ is reduced to the smallest cube containing all minterms not covered by any other cube of F. In contrast, Espresso-HF computes, for each $c \in F$, the smallest dhf-implicant containing all required cubes that are not covered by any other cube in F. Using this notion, the remaining steps are identical to the approach used in Espresso.

3.8 Make dhf-prime

The cover being constructed so far does not necessarily consist of dhf-primes. It is usually desirable to expand each cube of the cover to make it dhf-prime as a last step. This can be achieved by a modified expand step. The following greedy algorithm will expand an implicant c to a dhf-prime: While dhf-feasible, raise a single entry of c.

4 Existence

For certain Boolean functions and sets of transitions, no hazard-free cover exists. The exact hazard-free minimization method [9] is able to decide if a solution exists only after generating all dhf-prime implicants. A solution does not exist if and only if the dhf-prime implicant table includes at least one required cube not covered by any dhf-prime implicant.

Since the generation of all primes may very well be infeasible for even medium-sized examples, it is necessary to find an alternative approach. We present a new theorem for the existence of a solution leading directly to a simple algorithm.

Theorem 4.1 *A solution of the hazard-free minimization problem exists iff* $supercube_{dhf}(q)$ *is defined for all required cubes* q.

The proof is immediate from the discussion in Section 3.2.

Example. Consider Figure 5. To check for existence, we compute $supercube_{dhf}(q)$ for each required cube q. Except for abd, it holds that $q = supercube_{dhf}(q)$ since no privileged cube is intersected illegally. To compute $supercube_{dhf}(abd)$, note that privileged cube c is intersected illegally, i.e. $supercube_{dhf}(abd) = supercube_{dhf}(bd)$. Since bd now intersects privileged cube $a'c'$, we get $supercube_{dhf}(abd) = supercube_{dhf}(b)$ leading directly to the fact that $supercube_{dhf}(abd)$ does not exist because b intersects the off-set. Thus, there is no hazard-free cover for this example. □

5 Experimental Results

A prototype version of the Espresso-HF algorithm was run on several benchmark circuits on a SPARC 10 workstation, as shown in Figure 8. The results for the exact columns were obtained by running the exact hazard-free minimizer by Fuhrer/Lin/Nowick [2]. This method uses Espresso to generate all prime implicants, then transforms them into dhf-prime implicants, and finally employs MINCOV to solve the resulting unate covering problem. Each of the algorithms used in the three steps is critical, i.e. has a worst-case run-time that is exponential.

Three of the fifteen examples could not be solved by the exact minimizer (in the allotted time of 40 hours). For *stetson-p1* the generation of all prime implicants was not possible. For *cache-ctrl*, the exact minimizer was unable to transform the set of prime implicants into the set of dhf-prime implicants. For *pscsi-pscsi*, the resulting covering table was too large. This shows that the exact algorithm is not useful for large examples in general since all three steps can fail.

The heuristic minimizer Espresso-HF was able to solve all examples. For all but one example that could be solved by the exact minimizer, Espresso-HF finds the optimal solution. It is worth pointing out that many examples were very positively influenced by our notion of essentials. Quite a few examples can be minimized by just the essential step, resulting in a guaranteed minimal solution.

		exact[FLN]			**Espresso-HF**		
name	*i/o*	*#p*	*#c*	*time*	*#e*	*#c*	*time*
cache-ctrl	20/23	*	*	*	50	105	2136.87
dram-ctrl	9/8	45	22	0.15	22	22	0.24
pe-send-ifc	12/10	454	27	1.48	27	27	0.86
pscsi-ircv	8/7	20	12	0.07	12	12	0.06
pscsi-isend	11/10	204	23	0.43	23	23	0.39
pscsi-pscsi	16/11	65060	*	*	55	78	70.52
pscsi-tsend	11/10	190	22	0.41	22	22	0.48
pscsi-ts.-bm	11/11	188	23	0.46	23	23	0.54
sd-control	18/22	1718	34	40.85	23	36	15.05
sscsi-is.-bm	10/9	87	22	0.20	22	22	0.49
sscsi-tr.-bm	10/9	113	24	0.22	21	24	0.54
sscsi-ts.-bm	11/10	93	20	0.25	20	20	0.41
stetson-p1	32/33	*	*	*	34	62	447.33
stetson-p2	18/22	1574	37	34.29	26	37	15.85
stetson-p3	6/4	10	7	0.05	7	7	0.02

Figure 8: Comparison of exact and heuristic minimization (#p - number of dhf-prime implicants, #c - number of cubes, #e - number of essential equivalence classes, time - run-time in minutes)

The detection of essentials is crucial for speed and size. Espresso-HF currently exists only as a prototype. Run-times are expected to improve significantly by careful implementation.

6 Conclusions

To the best knowledge of the authors, the presented algorithm Espresso-HF is the first heuristic method based on Espresso to solve the hazard-free minimization problem for multiple-input changes.

Our prototype can solve all examples that we have available so far, and almost always obtains an absolute minimum-size cover. This includes examples that have not been solved before.

Espresso-HF overcomes the three bottlenecks of the exact method—prime implicant generation, transformation of prime implicants to dhf-prime implicants, and solution of the covering problem—each of which being solved by an algorithm with exponential worst-case behavior.

Espresso-HF also employs a new and much more efficient algorithm to check for existence without generating all prime implicants.

References

[1] R.K. Brayton et al. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic, 1984.

[2] R. Fuhrer, B. Lin, and S.M. Nowick. Symbolic hazard-free minimization and encoding of asynchronous finite state machines. In *ICCAD-1995*.

[3] A. Marshall, B. Coates, and P. Siegel. The design of an asynchronous communications chip. *Design and Test,* June 1994.

[4] G. De Micheli, R. K. Brayton, A. Sangiovanni-Vincentelli. Optimal state assignment for finite state machines. *IEEE TCAD,* CAD-4(3):269–285, July 1985.

[5] S.M. Nowick and D.L. Dill. Synthesis of asynchronous state machines using a local clock. In *ICCD-1991*.

[6] S.M. Nowick and B. Coates. Automated design of high-performance unclocked state machines. In *ICCD-1994*.

[7] S.M. Nowick, M.E. Dean, D.L. Dill, and M. Horowitz. The design of a high-performance cache controller: a case study in asynchronous synthesis. In *HICSS-1993*.

[8] S.M. Nowick, N.K. Jha, and F. Cheng. Synthesis of asynchronous circuits for stuck-at and robust path delay fault testability. In *VLSI Design 1995*.

[9] S.M. Nowick and D.L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE TCAD,* CAD-14(8):986–997, August 1995.

[10] R. Rudell and A. Sangiovanni Vincentelli. Multiple valued minimization for PLA optimization. *IEEE TCAD,* CAD-6(5):727–750, September 1987.

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

[12] K. Yun and D.L. Dill. A high-performance asynchronous SCSI controller. In *ICCD-1995*.

[13] K. Yun, D.L. Dill, and S.M. Nowick. Synthesis of 3D asynchronous state machines. In *ICCD-1992*.