An Efficient Divide and Conquer Algorithm for Exact Hazard Free Logic Minimization

J.W.J.M. Rutten, M.R.C.M. Berkelaar, C.A.J. van Eijk, M.A.J. Kolsteren
Eindhoven University of Technology
Information and Communication Systems Group
e–mail: jeroen@ics.ele.tue.nl

Abstract

In this paper we introduce the first divide and conquer algorithm that is capable of exact hazard–free logic minimization in a constructive way. We compare our algorithm with the method of Dill/Nowick, which was the only known method for exact hazard–free minimization. We show that our algorithm is much faster than the method proposed by Dill/Nowick by avoiding a significant part of the search space. We argue that the proposed algorithm is a promising framework for the development of efficient heuristic algorithms.

1 Introduction

One of the bottlenecks of asynchronous design is the logic synthesis step. During this step it is often required that the logic expressions to be generated are free of hazards for a given set of transitions under the unbounded wire delay model [1]. This is e.g. the case during the synthesis of Asynchronous Burst Mode Machines. It has been shown that these machines can successfully be used to implement large asynchronous designs [4]. Until recently hazard–free logic expressions had to be designed by hand, since no algorithm was known to generate them. The first algorithm capable of generating hazard–free logic expressions was proposed by Dill/Nowick [1]. This algorithm is capable of generating an exact pla–based solution. It uses an exact two–level minimizer, like espresso [5], to generate all prime implicants. A filter is applied to all these prime implicants to render them dynamic hazard–free (dhf). A minimal selection is made of the fittest implicants to generate an exact minimal solution in terms of the number of prime implicants. Since there can be an exponential number of prime implicants, this method is only applicable to relatively small examples.

In this paper we propose a new exact divide and conquer algorithm, similar to the one used in espresso, that is capable of generating hazard–free logic expressions in a constructive way. That is, we directly generate hazard–free logic without the need of a–posteriori fixing. Not only is this very important from a theoretical point of view, since we believe it is the first in its kind, it is also important from a practical point of view since by some modifications the algorithm is capable of generating both exact and heuristic solutions. We show that this algorithm can avoid part of the search space and therefore is much faster than Dill/Nowick's method for larger examples. In this paper we focus on single output functions. Single output minimization is an important operation during multi–level logic synthesis and the described algorithms can be applied in that field. We also outline the modifications necessary to allow the described algorithms to operate on multiple output functions; due to lack of space details are omitted.

2 Background

A binary–valued input, binary–valued output function can be described by $f : B^n \rightarrow B \cup \{*\}$ where $B = \{0, 1\}$. Here, a star indicates the part of the function belonging to the don't care set. The offset of f is denoted as R.

A cube S is a non empty Cartesian product space $S_1 \times S_2 \times ... \times S_n$, where $S_i \subseteq B$. A minterm is a cube where $|S_i| = 1$. An implicant is a cube that has no intersection with the offset. A prime implicant is an implicant that is not covered by any other implicant. Given a function f, the goal of 2–level minimization is to find a minimal selection of prime implicants such that the on–set of this function is completely covered.

Let a function f be expressed over the input variables $x_1 ... x_n$. $f|_{x_i}$ is called a cofactor of function f with respect to input variable x_i. It is defined as: $f|_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n)$. The negative cofactor, $f|_{\overline{x_i}}$ is defined as: $f|_{\overline{x_i}} = f(x_1, \ldots, x_i = 0, \ldots, x_n)$.

A function is said to be unate in a variable x_i if $f|_{x_i} \subseteq f|_{\overline{x_i}}$ or $f|_{\overline{x_i}} \subseteq f|_{x_i}$. In the first case the function is positive unate in x_i, in the latter case, the function is said to be negative unate in x_i. A function is unate if it is unate in each variable.

3 A new divide and conquer technique

In this section we introduce a new divide and conquer algorithm, the *threeway method*, that is capable of calculating all prime implicants. We introduce this algorithm because it can be converted to generate the set of all dhf–prime implicants,

which, to the best knowledge of the authors, is not possible with e.g. the algorithm used by espresso. Furthermore, we show that this algorithm is capable of avoiding a significant part of the search space by only generating contributing dhf–prime implicants. We start by introducing the problem of hazard–free logic minimization.

3.1 Hazard free logic minimization

Hazard–free logic is based on the dynamic hazard–free (dhf) implementation of a set of transitions. A transition is a tuple defined as: $< I, O >$ where $I \in D^n$ and $O \in D^m$. D is given by: $D = \{0, 1, \uparrow, \downarrow\}$. A transition cube [1] is a cube notated as [*A, B*], where A and B are minterms that indicate the starting–point and end–point of a transition in the input space. The transition cube is the smallest cube that covers both A and B. Hazard–free logic considers each transition individually: each transition is implemented free from hazards. At the start of each transition, the logic implementing the behavior of the transition is assumed to be stable.

Hazards can be static or dynamic [1]. Static hazards can be avoided by demanding that the transition cube is covered by an implicant in the solution. Such a transition cube is called a *required cube*. Dynamic hazards are taken care off by so–called *privileged cubes*. A privileged cube is a transition cube with an annotated starting–point. It is formally denoted as: $p = (p^c, p^s)$, where p^c is called the body of the privileged cube and p^s is called the starting–point. All implicants in a dhf–solution are only allowed to intersect a privileged cube when the starting–point is also included. Otherwise the intersection is said to be illegal. An implicant that does not illegally intersect any privileged cube is called a dhf–implicant. A dhf–prime implicant is not covered by any other dhf–implicant. A minimal dhf–solution consists of a minimal selection of dhf–prime implicants that covers all required cubes [1].

Example 1 Consider a circuit with 4 inputs and 1 output that is supposed to implement the following transitions free from dynamic hazards: {<↑000,1>, <1↑0↑,↓>, <11↑1,0>, <111↓,↑>, <1↓10,↓>, <↓010,0>, <00↓0,↑>}. The order of the input variables in these transitions is abcd. Furthermore we assume that the circuit implementing all transitions is stable at the start of each transition. In figure 1 a Karnaugh map is depicted showing all the transitions and the on/offset that can be derived from this set of transitions. An empty entry corresponds with a don't care entry.

If we use a 2–level minimizer, like espresso, to generate a minimal set of prime implicants that covers the depicted onset, the solution would be: –1–0, –00–. This solution is hazardous, which can be shown by examining transition <1↑0↑,↓>. In this transition, input variables b and d turn on. Suppose input variable b turns on before input variable d.

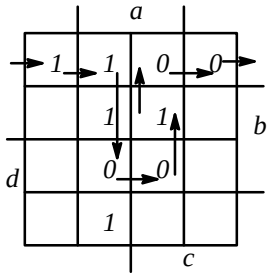

Figure 1: Karnaugh map for a given set of transitions

We can observe that cube –00– will turn off. Cube –1–0 will turn on, so we have a term takeover that can give rise to a static hazard: the output might temporarily turn off. This hazard can be avoided by demanding that the solution covers the smallest cube that covers both the starting–point of the transition and the minterm entered by turning b on. Since a solution must cover this cube, it is called a required cube. Here, the required cube is: 1–00. Let us add this required cube to the solution.

Now let us examine if we have successfully removed all hazardous situations for the given transition. Again, we assume that input variable b is the first to turn on. Because of the added required cube, the output will remain at one. Now input variable d switches on. Both the added required cube and cube –1–0 1 will turn off in that case. However the behavior of this cube might be observed after the required cube itself has turned off. In fact, the behavior of cube –1–0 going *on* might be observed after the required cube has turned off, due to our unbounded wire delay model. This might lead to a dynamic hazard: after the output has turned off, a glitch representing the behavior of cube –1–0 might be observed. This dynamic hazard can be avoided by demanding that during a transition after which the output will turn off, no cube can temporarily turn on. A cube may turn on, or it may turn off, but not both. This constraint is met by taking into account a set of privileged cubes. The body of a privileged cube is equal to the transition cube that covers the transition. A privileged cube also has an annotated starting–point, corresponding to the starting–point of the transition. A set of implicants is said to be dhf if the intersection of each implicant with the body of a privileged cube also contains the starting–point of that privileged cube. If this is not the case, the implicant is said to intersect a privileged cube illegally. This can lead to a dynamic hazard.

In this example, the privileged cube necessary to avoid a dynamic hazard during transition <1↑0↑,↓> is (p^c = 1–0–, p^s = 1000). Observe that prime implicant –1–0 intersects the body of this privileged cube, but does not cover the starting–point. Therefore this prime is not dynamic hazard free. For this example a correct minimal hazard free

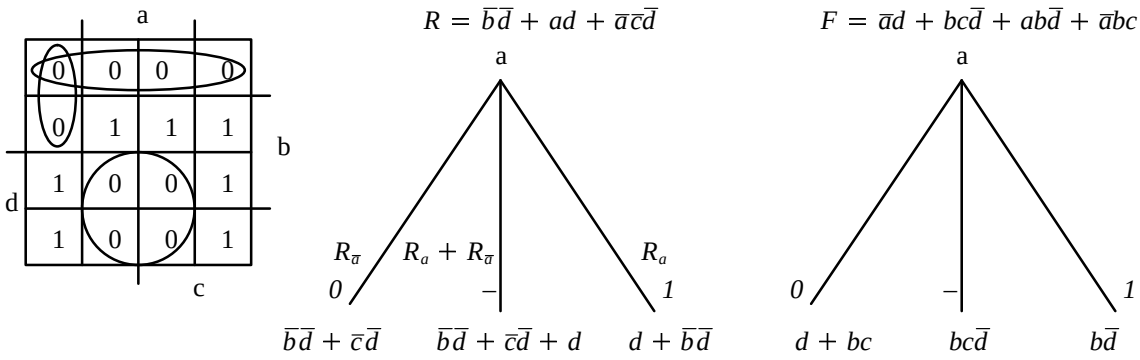

Figure 2: Example of the threeway method

solution consists of three dhf–prime implicants namely: –110, ––00, –00–.

3.2 The threeway method

In the threeway method for binary–valued functions, the set of all prime implicants is calculated by supplying the method the offset of a function *f*. Searching the set of all prime implicants becomes equal to searching all maximal cubes that do not intersect the offset.

If we consider a variable with respect to a cube, we can discern three cases: the variable is 0, 1 or – (don't care). The main idea behind the threeway method is to divide the original problem of calculating all prime implicants into the three sub–problems of calculating the set of prime implicants in which a selected variable has one of these values. Hence the name threeway method. For example, one of the sub–problems is to generate the set of prime implicants for which the selected variable is equal to 1. The basic idea of the method is depicted in figure 2. The idea to partition the problem of calculating the set of all prime implicants into three sub–problems is not new, it was also used by Coudert to implement an efficient implicit 2–level minimizer [3].

In the example in figure 2, variable *a* is used to reduce the original problem of finding all prime implicants into the three sub–problems of finding the prime implicants for which variable *a* has one of the three possible values.

These three values are represented by three edges. The set of primes for which *a* is equal to 0 can never intersect with that part of the offset where a is equal to 1. Therefore in order to calculate the set of primes for which *a* is 0, the zero–edge for future references, we only need to calculate the set of maximal cubes that have no intersection with the negative cofactored offset, since we abstract from variable a. This is expressed as $R|_{\bar{a}}$. For the one edge, the offset becomes $R|_a$. For the don't care–edge, the offset becomes $R|_a + R|_{\bar{a}}$. This is because for any prime for which variable *a* is a don't care, an intersection with $R|_a$ or $R|_{\bar{a}}$ implies an intersection with *R* itself and viceversa. The new sub–problems are reduced: the cardinality of the support set is decreased.

The original problem can be divided into new sub–problems repeatedly until the offset becomes empty or equal to the universe cube. In the first case, the set of primes becomes equal to the universe cube, since this is the maximal cube that does not intersect the empty offset. In the second case the set of primes is empty. The selection of edges taken, i.e. the path, determines the assignment of the variables of the prime.

For the given example in figure 2, the offset is given by $R = \bar{b}\bar{d} + ad + \bar{a}\bar{c}\bar{d}$, the cofactors are $R|_a = \bar{b}\bar{d} + d$ and $R|_{\bar{a}} = \bar{b}\bar{d} + \bar{c}\bar{d}$. By repeating the splitting process, the set of primes not intersecting $R|_a$ can be calculated. This set turns out to be equal to $Primes_a = b\bar{d}$. Similarly the set of primes not intersecting $R|_{\bar{a}}$ is equal to $Primes_{\bar{a}} = d + bc$. Also it turns out that the set of primes not intersecting $R|_a + R|_{\bar{a}}$ is equal to $Primes_{a+\bar{a}} = bc\bar{d}$. Combining these primes results in:

$Primes = a \cdot Primes_a + \bar{a} \cdot Primes_{\bar{a}} + Primes_{a+\bar{a}} =$
$ab\bar{d} + \bar{a}d + \bar{a}bc + bc\bar{d}$.

Figure 3 gives a general overview of the proposed threeway method:

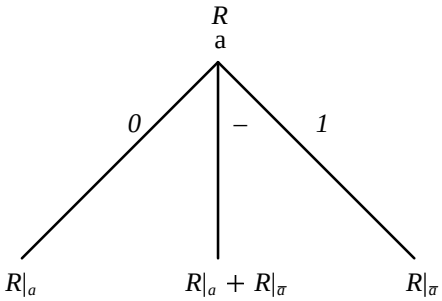

Figure 3: General threeway model

The primes generated by the zero– or one–edge can be covered by the primes generated by the don't care–edge. Therefore a single cube containment (SCC) filtering has to be performed, to remove all covered cubes, while merging the solutions of the sub–problems into the solution of the original problem. This is expressed as:

$Primes = SCC(a \cdot Primes_a + \bar{a} \cdot Primes_{\bar{a}} + Primes_{a+\bar{a}})$

We will now prove that the proposed method generates all prime implicants. In order to do so, we need the following lemma:

Lemma 1 Each cube has an unique path in the search tree. No two different cubes follow the same path in the search tree since this would imply that for each input variable they are assigned the same values.

Theorem 1 *Each cube generated by the threeway method is an implicant.*

Proof: We only generate a cube when the offset becomes empty. Certain paths exist in the search tree after which the offset becomes empty. By lemma 1 each path corresponds to a unique cube. Following a path is equal to calculating the cofactor of the offset with respect to this cube. Since this cofactor is empty, the cube must be an implicant. □

Theorem 2 *Each implicant generated by the threeway method is prime.*

Proof: Suppose prime implicant d covers an implicant c generated by the threeway method. Since d does not intersect the offset, the cofactor of the offset w.r.t. d is empty. Therefore, following the path that belongs to d leads us to a leaf–node where the offset is empty. So d is created. Implicant c might also be created, but due to the SCC–filter, it is removed. □

Theorem 3 *The threeway method will generate all prime implicants.*

Proof: Suppose a prime d is missing. Again we can follow its path and observe that the offset again will be empty. Therefore d will be generated. □

Theorem 4 *If the offset is unate in variable a, then the threeway method can be reduced to a twoway method.*

Proof: The primes generated by $Primes_a$ or $Primes_{\bar{a}}$ can be covered completely by the primes of $Primes_{a+\bar{a}}$. This happens when the offset, and therefore also the onset is unate in a certain variable. If the offset is unate in variable a then $R_a \subseteq R_{\bar{a}}$ or $R_{\bar{a}} \subseteq R_a$. In that case $R_a + R_{\bar{a}} = R_{\bar{a}}$ or $R_a + R_{\bar{a}} = R_a$. Therefore the primes generated by the zero–(one–) edge will be covered completely by the primes generated by the don't care–edge. Therefore the threeway method becomes a twoway method. □

We can use this observation to significantly reduce the search space by selecting unate variables as soon as possible.

3.3 Generating all dhf–prime implicants with the dhf–threeway method

Here, we propose modifications that will allow the threeway method to generate the set of all dhf–prime implicants. This is accomplished by keeping track of the set of privileged cubes during the process of splitting the original problem of calculating all dhf–primes.

The problem of calculating the set of all prime implicants is transformed into one of calculating all dhf–prime implicants. For the sub–problems we now want to calculate the set of all *dhf–primes* for which the splitting variable a assumes values zero, one or don't care.

Let us examine the set of dhf–prime implicants for which variable a is equal to 1. Since variable a is equal to 1 for any prime calculated by the 1–edge, any privileged cube that has a starting–point for which a is equal to 0, will be intersected illegally when its body is intersected. To avoid this intersection we must add the cofactored body, p_a^c of each of these privileged cubes to the offset of the 1–edge. This way, we can guarantee that the set of primes generated by the 1–edge will not intersect these privileged cubes illegally. However, the primes still might intersect those privileged cubes that have a starting–point for which variable a is equal to one or don't care. Therefore, we must keep track of these privileged cubes by passing them on. Resuming, the offset and set of privileged cubes that are passed on to the 1–edge are expressed as:

$$P_1 = \left\{ p|_a \mid p|_a^s \neq \emptyset \right\}$$
$$R_1 = R|_a + \left\{ p|_a^c \mid p|_a^s = \emptyset \right\}$$

We represent the set of privileged cubes and the Offset passed on to the 1–edge by adding a subscript 1.

We now consider the case where the splitting variable a is equal to don't care: we follow the don't care–edge. We can not determine which subset of the set of privileged cubes will always be intersected illegally. Therefore we must consider the set of all co–factored privileged cubes. This is expressed as:

$$P_2 = P|_a + P|_{\bar{a}}$$

The offset is expressed as:

$$R_2 = R|_a + R|_{\bar{a}}$$

A subscript of 2 represents the sets that are passed on to the don't care–edge. Note that P_2 is not equal to $P_0 + P_1$. In figure 4 the general model of the dhf–threeway method is depicted.

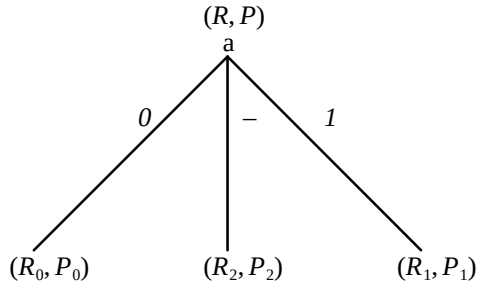

Figure 4: General model of the dhf–threeway method

Theorem 5 *The set of implicants generated by the dhf–threeway method is dynamic hazard free.*

Proof: By contradiction. Suppose the method generates an implicant c that illegally intersects a privileged cube p. We now start to follow the path belonging to c. After an edge, p can still be available, in a cofactored form, or it is removed during the process of calculating the set of privileged cubes that must be passed on. In the latter case, the cofactored form

of p^s is empty in which case the cofactored body of p, p^c is added to the offset. This would make an illegal intersection of c with p impossible. Since c is supposed to intersect privileged cube p illegally, p must remain in the transported set of privileged cubes. We can apply this story in a recursive way until the offset becomes empty. Here c is being created and it is equal to the universe cube. However, since c is equal to the universe cube, it cannot intersect p illegally, because the starting–point is included in the intersection. Therefore c is a dhf–implicant. □

Theorem 6 *The set of implicants generated by the dhf–threeway method is dhf–prime.*
Proof: Suppose d is a dhf–implicant that covers dhf–implicant c generated by the threeway method. Now let's follow the path belonging to d. The threeway method didn't generate d, meaning that when we trace d we end up at a leaf–node where the offset is not empty. We assume that d does not intersect the original offset. The offset, intersected by d, therefore is due to the body of at least one privileged cube that has been added to the offset. This implies that d will at least illegally intersect one privileged cube. Therefore d cannot be a dhf–prime implicant. □

Theorem 7 *The set of dhf–primes generated by the dhf–threeway method is complete.*
Proof: The threeway method generates all prime implicants. The dhf–threeway method generates dhf–prime implicants. Suppose a dhf–prime c is missing from the set generated. If we follow the path belonging to this prime we will discover that at the node where c should have been created, the offset is not empty. Therefore c is not valid: it intersects the original offset or it intersects a privileged cube illegally. The set of dhf–primes is complete. □

Again we can reduce the threeway method to at most a twoway method if the offset is unate in at least one variable. However, in the dhf–threeway method this constraint is not enough. The set of privileged cubes must also be taken into account since these privileged cubes can eventually contribute to the offset in the process of calculating the set of dhf–primes for one of the sub–problems. Therefore, we can only skip the 0 (1) edge if P_2 is equal to P_0 (P_1) and $R_2 = R_0$ ($R_2 = R_1$).

3.4 Generating contributing (useful) dhf–prime implicants

The dhf–threeway method is capable of generating all dhf–prime implicants. However, many of these dhf–primes will never contribute to a valid solution because they only cover the don't care set or because they only partially cover some required cubes. The threeway method for binary functions can be modified to avoid the generation of these primes. This is possible because the solutions of sub–problems do not interact to generate the solution of the original problem, as is the case in the unate recursive paradigm used in espresso. This means that if we know that an edge will not generate a set of contributing/useful primes, we can just discard that edge. It will have no influence on the other primes being generated.

In order to detect edges that will not generate contributing primes we also must keep track of the set of required cubes. The idea is to refrain from calculating the primes belonging to a certain edge when the set of required cubes is empty. In figure 5 the modifications necessary to the dhf–threeway method are depicted.

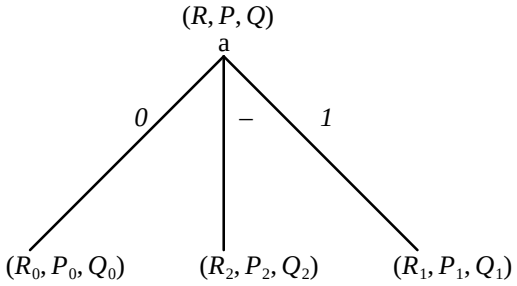

Figure 5: General model to generate contributing dhf–primes

Here Q is used to represent the set of required cubes. We already know how to calculate sets R_k and P_k. What remains to be answered is how to determine sets Q_k. Let us start by examining the 1–edge. We only want to propagate those required cubes in Q for which variable a also is equal to 1. All other required cubes are only partially covered, or not covered at all. So we can express Q_1 as:

$$Q_1 = \{q|_a \mid a \cdot q = q\}$$

A similar expression exists for Q_0. For the don't care–edge we must propagate all, cofactored, required cubes. So we can express Q_2 as:

$$Q_2 = Q|_a + Q|_{\bar{a}}$$

3.5 Multiple output functions

In this section we will discuss in an outline how the described threeway method can be modified to take into account multiple output functions. In order to do so, we have extended the threeway method to be able to deal with multiple–valued functions. All output variables can be combined into one multi–valued variable with a cardinality equal to the number of outputs [5]. This multi–valued variable can be used to split the original problem into three new sub–problems, just like a binary variable. This is done by partitioning this multi–valued variable into two partitions. For the 0 and 1–edge the offset simply become equal to the offset cofactored to the appropriate partition. This, however, is not the case for the don't care–edge. It turns out that the offset for the don't care–edge is equal to the combined sets of both cofactors of the offset, where the original multiple–valued variable is split into two new multiple–valued variables. Each new multiple–valued variable corresponds to one of

the partitions of the original multiple–valued variable. Due to lack of space we will not discuss the specific details nor the modifications necessary to modify this method to allow it to generate the set of all dhf–prime implicants. It will be the subject of a future paper.

4 Results and Conclusions

We have implemented the described threeway method and compared it with Dill/Nowick's method [1]. In order to do so, we used the two–level minimizer espresso to generate the set of all prime implicants. These primes are then rendered free from hazards by Dill/Nowick's hazard removal algorithm. In Table 1 the results are depicted. All benchmark circuits in the first column are derived from well known Asynchronous Burst Mode Machines and represent the biggest single output examples available. The number of inputs and the number of cubes of each circuit are shown in the second column. In the third column the number of primes generated by espresso and the runtime in seconds are given. A dash (–) indicates that the runtime was too small to measure with much accuracy (less than 0.1 [s]). The fourth column shows the number of dhf–prime implicants generated by Dill/Nowick's algorithm, which uses the results generated by espresso, and the runtimes. The fifth column represents the results by our threeway method when it is used to generate all dhf–prime implicants. The number of primes in the fifth column therefore must be equal to the number of primes in the fourth column. Again, the runtimes are also represented. The sixth column represents the number of primes and the runtime by the threeway method when it only produces the set of contributing dhf–prime implicants. The last column represents the smallest solution, in terms of dhf–primes, possible. This solution was obtained by solving a unate covering problem, where the minimum number of dhf–primes was selected that covers all required cubes.

All benchmark runs were performed on a HP9000/K260 system. Espresso and the threeway method prototype were both compiled with the same compiler options.

From table 1 it can be observed that the runtimes of the threeway method are comparable with the runtimes of espresso, when the times spend in the column hazard filter are added. However, the runtimes of the threeway method are significantly reduced when it is allowed to only generate the set of contributing prime implicants. This is caused by the significant reduction of the search space and of the number of primes calculated.

In this paper we have presented what we believe is the first constructive exact method that is capable of calculating the set of all dhf–prime implicants. We have shown that this method can easily be enhanced to generate only the set of useful primes. This significantly reduces the solution search space and hence the runtimes necessary. The proposed method can also be extended to deal with multiple output functions. To cope with larger practical problem instances, it is also important to search for efficient heuristic methods [2], especially when dealing with multiple–output functions. In a future paper we will show that the threeway method also provides a good basis for developing efficient heuristic methods.

References

[1] Steven M. Nowick, Davil L. Dill, "Exact Two–Level Minimization of Hazard–Free Logic with Multiple–Input Changes", ICCAD'92, pp. 626–630

[2] Michael Theobald, Steven M. Nowick, Tao Wu, "Espresso–HF: A Heuristic Hazard–Free Minimizer for Two–Level Logic", DAC'96, pp. 71–76

[3] Olivier Coudert, "Two–level logic minimization: an overview", Integration the VLSI journal, Vol. 17 No. 2, 1994, pp. 97–140

[4] K.Y. Yun, "Synthesis of Asynchronous Controllers For Heterogeneous systems", Stanford University, 1994, Ph.D. Thesis

[5] R. Rudell, A Sangiovanni–vincentelli, "Multiple–Valued Minimization for PLA Optimization", IEEE Transactions on computer–aided design, Vol. CAD–6, No. 5, September 1987

Table 1: results

Name	I/C	Espresso	Hazard Filter	dhf–threeway	Contrib	Minimize
sbuf–send–ctl.s.pla	6/50	8/–	10/–	10/–	4/–	3
isend–bm.s.pla	9/91	10/–	10/–	10/–	10/–	3
pe–send–ifc.s.pla	8/59	17/–	17/–	17/–	12/–	6
isend.s.pla	8/67	27/–	27/–	27/–	7/–	4
p2.s.pla	13/159	40/–	40/–	40/–	20/–	4
pscsi.s.pla	14/411	173/0.10	179/–	179/0.40	71/0.22	18
p1.s.pla	20/396	473/0.69	527/0.11	527/1.13	169/0.40	12
cache–ctrl.s.pla	21/1334	469/1.12	543/0.16	543/1.40	95/0.63	16
scsi.s.pla	18/912	1958/4.55	2001/0.26	2001/3.17	414/0.6	14