OPTIMISTA: State Minimization of Asynchronous FSMs for Optimum Output Logic

Robert M. Fuhrer Steven M. Nowick*
Department of Computer Science
Columbia University
New York, NY 10027
{rmf,nowick}@cs.columbia.edu

ABSTRACT

The *optimal state minimization problem* is to select a reduced state machine having the best logic implementation over all possible state reductions and encodings. A recent algorithm, OPTIMIST [3], was the first general solution to this problem for synchronous FSMs. In this paper, we present the first solution for asynchronous FSMs.

This paper makes two contributions. First, we introduce OPTIMISTA, a new algorithm which guarantees optimum 2-level output logic for asynchronous FSMs. In asynchronous machines, output logic is often critical: it usually determines the machine latency. The algorithm is formulated as a binate constraint satisfaction problem, which is solved using a binate solver. The second contribution is a novel alternative result: the unreduced machine itself can be used directly to obtain minimum-cardinality output logic.

Thus, this paper presents two approaches: using OPTIMISTA, which simultaneously performs state and logic minimization; or using no state reduction (if output logic cardinality is of sole interest). Extensions for literal optimization, targetted to multi-level logic, are also proposed.

1 INTRODUCTION

The tremendous amount of research in sequential synthesis has produced strong results and fast and robust algorithms for individual synthesis steps, such as state minimization, state assignment, 2-level logic minimization, and multi-level transformation. However, the quality of these results may still be significantly below the global optimum. Specifically, the classical sequential synthesis trajectory typically reduces the problem's complexity by decomposing it into several distinct phases. Each phase constitutes a local optimization problem, separated by "blinders" from other phases. The result is then a set of locally-optimal solutions contributing to a globally sub-optimal process.

OPTIMIST [3] was one attempt to address this issue broadly for technology-independent logic synthesis, by attacking the problem of *optimal state minimization* for incompletely-specified synchronous FSM's. The problem is defined as selecting the reduced machine that has the best 2-level logic implementation over all possible state minimizations and encodings. To this end, OPTIMIST breaks with traditional approaches, by performing state minimization and state encoding concurrently, using a powerful new form of symbolic logic minimization.

In this paper, we extend the OPTIMIST algorithm, presenting the first solution to the optimal state minimization problem for *asynchronous machines.*

The paper makes two contributions. First, we introduce OPTIMISTA, a new algorithm for optimal state minimization of asynchronous FSMs. The algorithm produces a reduced state machine which has minimum-cardinality 2-level hazard-free output logic over all possible state reductions and state encodings. The method is formulated as a binate constraint satisfaction problem, which is solved by a binate solver [1]. Unlike the original OPTIMIST algorithm, which is exact only under an input encoding model [11], the new method is the first truly exact solution to the optimal state minimization problem that is independent of the encoding model.

"Output-targetted state minimization" is especially appropriate for asynchronous machines, because it addresses the key performance parameter in systems of asynchronous machines: output latency. In an asynchronous system, input-to-output latency often determines performance, since state changes are not bound to a clock period. In practice, state changes often are non-critical (see, e.g., [10]), and can safely proceed in parallel with the propagation and processing of output changes.

Second, this paper offers a novel alternative approach: the unminimized machine itself can always be used to obtain exactly minimum-cardinality 2-level output logic, over all possible minimizations and encodings. That is, *no state reduction is necessary*. This result indicates that state reduction can never improve (i.e., decrease) 2-level output logic cardinality. This alternative approach is only useful if the sole cost metric is minimizing logic cardinality (i.e., number of products). The drawback, unlike OPTIMISTA, is that states will not be reduced.

Finally, an extension to OPTIMISTA for literal optimization, targetted to multi-level logic, is also proposed.

*This research was funded in part by NSF Award CCR-97-34803.

2 Background

2.1 Burst-Mode Asynchronous State Machines

Burst-Mode Specifications. An asynchronous state machine allowing multiple-input changes can be specified by a form of state diagram, called a *burst-mode specification* [13] (see example in Figure 1). Burst-mode specifications, and variants, have been used for several recent asynchronous design methods [13, 18, 12]. Arcs are labelled with possible transitions, taking the system from one state to another. Each transition consists of a non-empty set of input changes (an *input burst*) and a set of output changes (an *output burst*). In a given state, when all the inputs in some input burst have changed value, the system generates the corresponding output burst and moves to a new state. Inputs within a given input burst may arrive in any order and at arbitrary times.

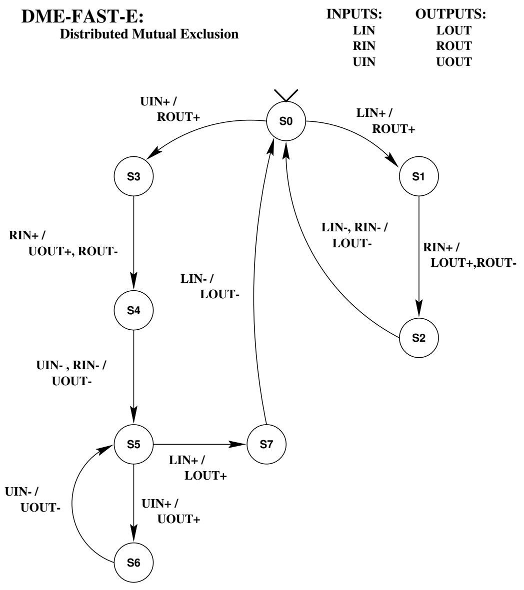

Figure 1: Example burst-mode specification.

Target Implementation. A burst-mode specification can be realized as a Huffman machine [17]. Initially, the machine is stable in some state. Inputs in a specified input burst then change value in any order and at any time. Throughout this input burst, the machine outputs and state remain *unchanged*. When the input burst is complete, the outputs change value monotonically as specified. A state change may also occur concurrently with the output change, and the machine will be driven to a new stable state. Alternatively, no state change may occur (if the destination state is merged with the source state, after state minimization). In either case, no further inputs may arrive until the machine is stable. That is, the machine operates in *fundamental mode* [17].

Figure 2 shows a fragment of the primitive (i.e., unminimized) flow-table for the burst-mode design of Figure 1, highlighting a single specified transition. A *horizontal transition* is the portion of the transition corresponding to the input burst. The *entry point* is the starting input column, and the *exit point* is the terminating input column of the horizontal transition. All points in the horizontal transition, exclusive of the exit point, are *stable points*, retaining the same output and next-state values as those at the entry point. A *vertical transition* corresponds to the state change portion of the transition, from an unstable state to a stable state.

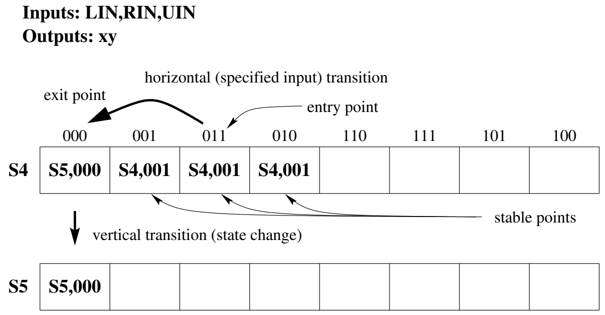

Figure 2: Horizontal and vertical transitions in a flow table

2.2 The State Mapping Problem

We assume basic background on state minimization [7, 8, 17]. We now review the state mapping problem, which also can have an impact on logic optimality.

M	0	1
s_0	s_0,0	s_2,0
s_1	s_1,0	s_1,-
s_2	s_1,-	s_0,1

M'	0	1
$s_0' = \{s_0, s_1\}$	s_0',0	s_1',0
$s_1' = \{s_1, s_2\}$	$\{s_1\}$,0	s_0',1

Figure 3: State table before and after minimization

Given an incompletely-specified FSM, a state reduction often defines a *set* of compatible realizations [16, 6]. In this case, the next-state behavior of the resulting ISFSM forms a *relation*, so that different next-state bindings are possible [9]. The state mapping problem is illustrated in Figure 3, taken from [9]. A choice of next-state in M' at total state $\langle\ 0\ s_1'\ \rangle$ exists; it can be assigned to either s_0' or s_1'.

2.3 Symbolic Hazard-Free Logic Minimization

Since we are concerned with hazard-free two-level minimization of output logic to be performed in a *symbolic domain* (i.e. given by a flow table), we briefly summarize results on multiple-valued-input (mvi) hazard-free logic minimization. For more details, see [2].

Definition 2.1 (Multiple-valued transition cube)
A **multiple-valued transition cube** *is a cube with a* **start point** *and an* **end point***. The multiple-valued transition cube, denoted as $[A, B]$, from A to B (in a multi-valued domain) has start point A and end point B and contains all minterms that can be reached during a transition from A to B.*

A function f which does not change monotonically during an input transition is said to have a *function hazard* in the transition (see [2]). If a transition has a function hazard, no multiple-valued implementation of the function is guaranteed to avoid a glitch during the transition, assuming arbitrary gate and wire delays. Therefore, we consider only transitions which are free of function hazards (cf. [14]). If f is free of function hazards for a transition from input A to B, an implementation may still have a **logic hazard** due to possible delays in the logic realization.

Definition 2.2 (Required and privileged cubes) *Given a multiple-valued function f, and a set, T, of specified function-hazard-free multiple-valued input transitions of f, every cube $[A, B] \in T$ corresponding to a static $1 \rightarrow 1$ transition, and every maximal subcube $[A, X] \subset [A, B]$ where f is 1 and $[A, B] \in T$ is a dynamic $1 \rightarrow 0$ (or $0 \rightarrow 1$) transition, is called a* **required cube**. *Furthermore, every cube $[A, B] \in T$ corresponding to a dynamic $1 \rightarrow 0$ transition is called a* **privileged cube**. *(A dynamic $0 \rightarrow 1$ transition from A to B will hereafter be considered as a dynamic $1 \rightarrow 0$ transition from B to A, with corresponding privileged cube $[B, A]$ and start point B.)*

Definition 2.3 (Multiple-valued DHF-implicants) *An implicant* **illegally intersects** *a privileged cube if it intersects the privileged cube but does not contain its start point. A* **multiple-valued DHF-implicant** *is an implicant which does not intersect any privileged cube of f illegally. A* **multiple-valued DHF-prime implicant** *is a multiple-valued DHF-implicant contained in no other multiple-valued DHF-implicant.*

A *hazard-free cover* of function f is a cover of f whose multi-valued AND-OR implementation is free of logic hazards for a *given set* of specified input transitions. Finally, the **two-level mvi hazard-free logic minimization problem** is *to find a minimum cost cover* of a function **using only multiple-valued DHF-prime implicants** where **every required cube is covered.**

It is well-known that, given an arbitrary function f and set T of function-hazard-free input transitions, a hazard-free cover *may not exist* [13]. Burst-mode sequential synthesis algorithms [13, 18] have a solution to this problem: by imposing sufficient constraints on state minimization, it is guaranteed that the resulting Boolean functions will *always* have a hazard-free cover [15, 12, 18]. Similar restrictions on state minimization must be addressed by our new algorithm as well.

3 Overview of OPTIMISTA

We now give an overview of the flow of our new algorithm. More details are presented in Section 4.

First, two sets of covering objects are generated from the original burst-mode specification. In particular, a set of *state compatibles* is generated, using the classic synchronous compatibility relation [6], followed by a filtering step. Then, an ordinary set of *DHF symbolic output prime implicants* is formed. These primes are formed *on the unreduced machine*. DHF implicants can be used to cover the reduced machine because of a relationship between implicants on the unreduced and reduced machines. *No next-state implicants are generated.*

Then, *binate covering constraints* are generated, using the given state compatibles and DHF primes. Covering constraints subsume those used in classic state minimization. They fall into four categories: classic state reduction requirements (like those in [6]), state mapping, hazard-free logic covering requirements for the outputs of the reduced machine, and constraints to ensure the existence of a hazard-free implementation for the next-state.

The *binate constraints are solved* using Scherzo [1], under a cost model that targets logic cardinality.[1] The result is a closed state cover, state mapping, and a selection of DHF symbolic primes having minimum cardinality over all valid state reductions.

Using the selected state compatibles, state mappings, and output primes, a symbolic cover is formed, or *"instantiated"*, on the reduced machine using a so-called "natural mapping". The result is a hazard-free logic cover for the outputs of the reduced machine.

At this point, the method is complete. The reduced machine can then be passed to a burst-mode optimal state assignment tool, such as *CHASM* [2], to produce an exactly-minimum cardinality hazard-free binary implementation.

4 The OPTIMISTA Algorithm

4.1 State Compatible Generation

The compatibility relation has an impact on the existence of hazard-free logic because state merges can introduce unavoidable logic hazards. The goal is to identify the set of compatibles that, when used, guarantee the existence of hazard-free logic. There are five conditions that imply the incompatibility of a state set. The first four relate to the existence of a hazard-free logic cover, and the last, to the burst-mode requirement that each stable state (in the unminimized flow table) must be mapped to a stable state (in the reduced table):

1. No hazard-free cover exists for a horizontal output transition
2. No hazard-free cover for a vertical output transition (over all possible state mappings)
3. No hazard-free cover exists for a horizontal next-state transition (over all possible state mappings)
4. No hazard-free cover exists for vertical next-state transition (over all possible state mappings)
5. Some stable state (unreduced machine) is mapped to an unstable one (reduced machine)

Of the five conditions, OPTIMISTA directly detects only one during generation of state compatibles: the first condition. This condition is readily detected without overly complicated analysis. Thus, we identify a set of restrictions that are necessary but insufficient, and use it to prune the set of

[1] Alternatively, literal count may be targetted under certain assumptions; see Section 6.

compatibles. The remaining conditions are folded into the covering constraints ("state mapping incompatibility"), so as to let the binate solver do the remaining analysis.

OPTIMISTA generates state compatibles using the standard synchronous pairwise incompatibility relation, and prunes the resulting set using Algorithm 1.[2]

Algorithm 1. `Compatible filtering.`

```
function filter_compatibles(C) {
 C' ← C;
 for each compatible c ∈ C' {
  for each transition t originating in
    unreduced state s ∈ c {
   for each horizontal required
     output cube r of t {
    if there is no DHF prime which
     contains r and spans all states in c {
     // No hazard-free cover for r exists
       C' ← C' − {c}; } } } } }
```

4.2 Symbolic Prime Implicant Generation

OPTIMISTA uses ordinary DHF output primes, generated on the *unreduced machine*, for symbolic hazard-free logic minimization. We show later that these primes can be "naturally mapped" onto the reduced machine, and that their image includes the entire set of DHF primes for each reduced machine.

4.3 Binate Constraint Generation

Constraint Variables. The following table describes each of the decision variables involved in the binate covering constraints, and shows their assigned cost in the covering problem.

var	description	cost
c_i	select compatible c_i for the state cover	0
δ_{t,c_i,c_j}	map exit point of trans t in reduced state c_i to c_j	0
p_i	select DHF prime p_i for symbolic logic cover	1

State Covering.

This set of constraints ensures that, for each unreduced state, some compatible is selected which covers it. These constraints are identical to the state covering constraints in [6].

$\forall$ unreduced states s of $\mathcal{M}$

$$c_{i_1} + c_{i_2} + \cdots + c_{i_n}$$

where $\{c_i\}$ is the set of compatibles that contain s.

State Mapping. This set of constraints ensures that the exit point of each specified transition in the reduced machine is state-mapped appropriately. This is accomplished by selecting for transition t in c_i a destination state $s'_j = c_j$ from the set of valid choices. A set of decision variables $\{\delta_{t,c_i,c_j}\}$ is associated with each specified transition t in reduced state c_i, one for each valid state mapping choice c_j for that transition's exit point.

$\forall$ compatibles c*For each candidate compatible/reduced state*

$$c \rightarrow \delta_{t,c,c'_1} + \delta_{t,c,c'_2} + \cdots + \delta_{t,c,c'_n}$$

where $\{c'_j\}$ are the compatibles which cover exitPoint(t). The clause is conditional on the selection of compatible c, since only then does the transition t in c exist in the reduced machine.

State Mapping Coherency. These constraints ensure that *at most one* state mapping is selected for each reduced transition. It is necessary, because the set of N state mapping choices $\{c_j\}$ for reduced transition t in c_i is "encoded" in N distinct binary decision variables $\{\delta_{t,c_i,c_j}\}$. Without the following constraints, more than one may be selected.

$\forall$ compatibles c*For each candidate compatible/reduced state*

$\forall$ specified transitions t of $\mathcal{M}$ such that $PS(t) \in c$

$\forall$ distinct pairs of state mapping variables

$\langle \delta_{t,c,c_a}, \delta_{t,c,c_b} \rangle$ for t in c_i

$$\overline{\delta_{t,c,c_a}} + \overline{\delta_{t,c,c_b}}$$ *Select at most one mapping for t in c*

where PS is the (present) state in M in which transition t lies.

Functional Covering. Functional covering constraints select a set of DHF primes which form a symbolic hazard-free covering of the outputs.

Horizontal Transitions. The required cubes for the outputs during horizontal transitions are completely defined, regardless of the state mapping choices. Hence, functional covering of these required cubes can be performed without regard to state mapping.

$\forall$ compatibles c *For each possible reduced state*

$\forall$specified transitions t of $\mathcal{M}$ such that $PS(t) \in c$

$\forall$ horizontal required cubes r of t

$$c \rightarrow p_1 + p_2 + \cdots + p_N$$

where $\{p_i\}$ is the set of DHF primes which a) contribute to the output, b) contain c, and c) contain r; M is the unminimized machine, and PS is the (present) state in M in which transition t lies.

Vertical Transitions. In this case, the state mapping selection determines the actual span of the vertical transition. If the next-state is mapped unstably, the required cube spans 2 states; otherwise, it spans only 1.

[2]This algorithm assumes the set of DHF output-primes on the unreduced machine are already available. Also, without loss of generality, we only show the analysis for a single output.

$\forall$ compatibles c *For each possible "reduced state"*
$\forall$specified transitions t of $\mathcal{M}$ such that $PS(t) \in c$
$\forall$state mapping choices $\delta_{t,c,c'}$
$\forall$ vertical required cubes r of t when the exit point of t is mapped to c'
$$\delta_{t,c,c'} \rightarrow p_1 + p_2 + \cdots + p_N$$

where $\{p_i\}$ is the set of DHF primes which a) contribute to the output, b) span both c and c', and c) contain r.

State Closure. This set of constraints ensures that the selected set of state compatibles, along with the chosen state mappings, is closed, in the classical sense.

$\forall$ variables δ_{t,c_a,c_b} *For each state mapping choice*
$\delta_{t,c_a,c_b} \rightarrow c_b$ *If c_b is chosen as the destination state, must select it as well*

State Mapping Incompatibility. The goal of this set of constraints is to guarantee the existence of a hazard-free logic implementation for the next-state.

Algorithm 2 below considers the hazard-free covering of *horizontal required cubes* lying in a selected state. The algorithm only applies to next-state logic. (Algorithm 1 already ensures that a hazard-free cover exists for each horizontal output required cube.)

The algorithm assumes an *input encoding formulation* [11] for the next-state, a model used in many CAD algorithms. Each distinct next state symbol is given its own unique binary variable, so that, effectively, next states are (temporarily) 1-hot encoded. For example, a horizontal transition from (stable state) S_1 to an unstable exit state S_2, consists of two distinct state transitions: from $1 \Rightarrow 0$ for the S_1 bit, and from $0 \Rightarrow 1$ for the S_2 bit. (For details, see [5].)

Algorithm 2: `Identifying incompatible state mappings (horizontal)`

identifyHorizontalIncompatibilities(){ // Unoptimized version
  for each compatible c (call it state s') {
    for all state mappings of the transitions in c {
      for each transition t in c {
        for each horizontal required cube r of t {
          // Try to cover r
          $p' := r$; // Start by using r itself
          while p' illegally intersects a priv cube of t' in s' {
            $p' := p'$ expanded to contain $start(t')$;
            // OFF-set(s') is defined wrt state mapping of s'
            if p' intersects OFF-set of s' {
            disallow this mapping set (generate a constraint);
              continue with the next state mapping;
            } } } } } } }

The algorithm proceeds as follows. First, the outermost loop selects a reduced row (a compatible). Then, *every* combination of state mappings for all of the specified transitions in this row are enumerated. For each combination, the required cubes of each specified transition are identified. A covering implicant is then formed for each required cube, if possible. Specifically, the procedure attempts to use the required cube to cover itself, and expands it as necessary to avoid illegal intersections with any privileged cubes. If at any point the product hits the OFF-set, it is no longer an implicant. When that happens, no hazard-free cover exists for that required cube. In this case, an incompatibility constraint is generated which outlaws that particular state mapping combination.

A similar but more complex algorithm is used to insure that each *vertical required cube* can be feasibly covered (not shown, due to space limitations); see [5] for details.

4.4 Binate Constraint Solution

Once all constraints are generated, the constraints are solved by a standard binate solver. The solution to the constraints results in a selection of state compatibles, a selection of state mappings for each specified transition in the reduced machine, and a minimum-cardinality selection of DHF output prime implicants. Our implementation uses the Scherzo solver [1].

4.5 Instantiation

Symbolic instantiation is the process by which selected DHF-PI's (formed on the unreduced machine) are transformed one-for-one into a 2-level symbolic cover of the reduced machine. Given an output DHF-PI, $p = \langle IN,\ PS,\ OUT \rangle$, with input and present state fields, contributing to some outputs "OUT", we define

$$p' = \text{Instantiate}(p) = \langle IN,\ PS',\ OUT \rangle$$

The input and output fields are unchanged by instantiation. The present state field, PS, of the original DHF-PI (formed in the unreduced machine) will be mapped so as to cover *all selected compatibles* whose states are completely contained in PS. For example, if PS contains {s1,s3,s5,s6}, and the reduced machine had selected compatibles (i.e. rows) {s1,s2}, {s3,s5}, {s1,s6}, and {s4,s5}, then the DHF-PI would be instantiated only to cover rows whose compatibles it fully contains: {s3,s5} and {s1,s6}.

It can be shown that this *natural mapping* preserves hazard-freedom. Also, for any DHF-PI formed on the reduced machine, there is a corresponding DHF-PI in the unreduced machine which "naturally maps" to it.

5 Optimality of the Unminimized Machine

This section demonstrates an interesting and useful theoretical result. A hazard-free output logic cover *on the unreduced machine* itself has *minimum cardinality* (i.e., fewest products) over all possible state reductions and encodings. A consequence of this result is that state reduction can *never further reduce* output logic complexity.

Although the unminimized machine always gives the optimum solution for output logic cardinality, it does not necessarily do so under more practical cost functions. For example, it is well-known that state reduction often helps simplify the next-state logic.[3] Thus, the advantage of the

[3] This is of course the classic motivation for state minimization.

OPTIMISTA algorithm in Section 4 is that can obtain a solution which minimizes *both* output logic and state cardinality, when a suitable cost function is used.

We briefly state the key lemmas and theorems. For detailed proofs, see [5]. The burden of the argument is in showing that every prime hazard-free cover of a given reduced machine $\mathcal{M}'$ has a corresponding cover for the unreduced machine $\mathcal{M}$ of identical cardinality that maps onto it.

Lemma 5.1 *To every DHF output prime implicant* p' *on some reduced machine* $\mathcal{M}'$ *there corresponds at least one DHF output prime implicant* p *on unreduced machine* $\mathcal{M}$ *which Naturally Maps onto* p'.

From the above, given any output logic cover Π' for some reduced machine $\mathcal{M}'$, we can construct, member-wise, a set of implicants (call it Π) on $\mathcal{M}$.

Lemma 5.2 Π *is a hazard-free output cover for* $\mathcal{M}$.

Theorem 5.1 *The unminimized machine* $\mathcal{M}$ *possesses a two-level hazard-free output logic cover which has minimum cardinality across all possible state minimizations and encodings.*

Proof: This follows from Lemma 5.2, since, for every hazard-free output logic cover Π' for any reduced machine $\mathcal{M}'$, we can construct a hazard-free output cover for $\mathcal{M}$ of identical cardinality. □

6 Literal Optimization

With a small modification, the above algorithm can be transformed to a restricted, but useful, algorithm for *exact literal minimization*.

Interestingly, for each DHF-PI, the number of *input literals* is precisely known, even though these DHF-PI's are formed before state minimization and state assignment. This holds, because inputs are binary, so the columns covered by a DHF-PI are invariant under the instantiation step.

The new problem that is solved exactly is: Find a state minimization which results in a hazard-free cover of output logic *with fewest input literals*. This problem is important for asynchronous synthesis, since the input-to-output paths are typically critical: non-critical present state literals can be factored out, into additional levels of logic.

A simple modification of the above algorithm can be used: in the binate constraint matrix, replace the unit cost of each product by a weight corresponding to the number of input literals it contains.

7 Experimental Results

Table 1 summarizes the experimental results. A limited set of trial runs was performed, targetted to product minimization. OPTIMISTA does as well as the burst-mode tool, MINIMALIST [4, 5], in each case, but no better. This suggests that either there is no room for improvement in these circuits, or that MINIMALIST'S state minimization happens to be doing particularly well. We anticipate that gains may come from larger designs, which offer more latitude for optimization. We also plan to run comparative experiments on literal optimization.

8 Conclusions and Future Work

This paper introduces the first optimal state minimization algorithm for asynchronous state machines. This new method precisely targets output logic, producing exactly minimum cardinality two-level hazard-free output logic over all possible state minimizations, state encodings, and logic minimizations. We also outline extensions to handle literal optimization. Further experiments are needed, on a variety of examples, to see if better results can be obtained.

References

[1] O. Coudert and J.C. Madre. New ideas for solving covering problems. In *DAC-1995*, pages 641–646, 1995.

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

[3] R.M. Fuhrer and S.M. Nowick. Optimist: State minimization for optimal 2-level logic implementation. In *ICCAD-1997*, 1997.

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

[5] Robert M. Fuhrer. Sequential optimization of asynchronous and synchronous finite-state machines: Algorithms and tools. Technical report, Columbia University, 1999. Ph.D. Thesis.

[6] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE TEC*, EC-14:350–359, June 1965.

[7] G. Hachtel, J.K. Rho, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Trans. on CAD*, CAD-13(2):167–177, February 1994.

[8] T. Kam, T. Villa, R.K. Brayton, and A. Sangiovanni-Vincentelli. A fully implicit algorithm for exact state minimization. In *DAC-1994*.

[9] B. Lin and F. Somenzi. Minimization of symbolic relations. In *ICCAD-1990*, pages 88–91, 1990.

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

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

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

[13] S.M. Nowick and D.L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *ICCAD-1991*.

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

		compats			OPTIMISTA			
design	**i/s/o**	prime	all	DHF-PI's	cons	ms	**prods**	sec
dme-e	3/8/3	5	28	9	2122	†	-	-
dme-fast-e	3/8/3	4	20	11	694	9	6	8
pscsi-ircv	4/6/3	3	11	10	201	7	8	1
pscsi-trcv	4/6/3	3	11	8	180	10	6	1
pscsi-tsend	4/10/3	6	15	16	187	9	10	1
pscsi-tbm	4/10/4	6	14	11	171	13	9	2
sbuf-read-ctl	3/7/3	3	19	11	807	†	-	-

[‡]Constraint solution failed to produce a solution in a reasonable amount of time

Table 1: Experimental results for OPTIMISTA

[15] Steven M. Nowick. Automatic synthesis of burst-mode asynchronous controllers. Technical report, Stanford University, 1993. Ph.D. Thesis.

[16] M. Paull and S. Unger. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. on Elec. Comp.*, EC-8:356–367, Sept. 1959.

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

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