Recurrence Equations and the Optimization of Synchronous Logic Circuits.

Maurizio Damiani Giovanni De Micheli

Center for Integrated Systems
Stanford University

Abstract

In this paper we present a formulation for the problem of optimizing synchronous logic across register boundaries. We describe the degrees of freedom (i.e. the don't care conditions) of an embedded subnetwork by means of **sets of execution traces**, *described implicitly by Synchronous Recurrence Equations. The optimization problem reduces to that of finding minimum-cost solutions to such equations. An exact solution algorithm for this problem is presented, along with approximations that improve its computational efficiency. Eventually, we demonstrate the feasibility and effectiveness of the approach on synchronous benchmark circuits.*

1 Introduction.

Synthesis and optimization problems for combinational and synchronous logic circuits are currently the object of intense investigation. Boolean methods for combinational networks [1, 2, 3, 4, 5, 6, 7, 8] have matured both from a theoretical and application viewpoint. Such methods allow incomplete specification of the global behavior of a network, and work directly on its structural, hierarchical representation. They improve iteratively on the initial network by extracting subnetworks to be optimized and identifying precisely the degrees of freedom in their desired terminal behavior. Two-level optimization algorithms are then used for their optimization. In particular, in the case of single-output subnetworks, it has been proved that such degrees of freedom are fully represented by a *don't care set* [2].

By contrast, synthesis and optimization of synchronous circuits have so far relied on procedures based on (possibly iterative) manipulations of their *behavioral models*, typically in terms of *state transition graphs* [12, 13, 9, 10, 14, 11].

There are several disadvantages associated with this approach. First, a state diagram description does not allow us to represent any degree of freedom in the global behavior of a synchronous circuit. It is indeed often the case (see, for example, the control synthesis problem from high-level specifications [19, 20]) where the cycle-by-cycle behavior of a synchronous machine is only incompletely specified, and several *not equivalent* finite-state machines may meet the specifications. The second drawback of this approach is the remoteness of the state diagram model from the final implementation, that makes it difficult to evaluate the key figures of merit, such as area, testability and performance, during the optimization process.

It is then desirable to have available methods that, similarly to the combinational case, can optimize synchronous circuits directly from *structural* models, *i.e.* netlists, (according to some given area/performance metrics) and can take into account incomplete specifications [15, 24].

We show in this paper that, unlike the combinational case, capturing the degrees of freedom (the " *don't care conditions*") associated to a synchronous subnetwork entails being able to reason about *sets of sequences* of values in the synchronous network. This capability would allow the automation of logic transformations otherwise impossible. The following example is taken from [25]:

Example 1. The circuit shown in Fig. (1) computes a second order-moment M of the luminance of an image:

$$M = \sum_{i,j} i\ j\ a_{ij}$$

where a_{ij} is the luminance of the pixel of coordinates i, j. The binary counters C_i, C_j scan by rows the space of coordinates (i, j). C_j is clocked faster than C_i, and the product $p = i\ j$ is first computed by the multiplier M_1. Its output feeds a second multiplier M_2 along with the input a_{ij} to compute the term $i\ j\ a_{ij}$. These terms are eventually added up in the accumulator register *acc*. M_1 cannot be optimized by any combinational method. All input combinations are, in fact, asserted at its inputs by the two counters, so that ther are no external *controllability don't cares* . Moreover, any change at its outputs would be revealed by an error in the final result, and therefore we have no external *observability don't cares* .
It is possible, however, to optimize M_1 by observing that its input signals are fed in a precise order by the counters. Since during the scan of a row only C_j is incremented, the outputs of M_1 at two consecutive time-points, n and $n+1$, are *related* by the recurrence:

$$p_{n+1} = i\ (j+1) = i\ j + i = p_n + i$$

suggesting the simpler realization shown in Fig. (2). This optimization would have been impossible without knowledge of the recurrence. □

In this paper we consider first a behavioral representation of a synchronous circuit in terms of *execution traces*, as these allow us to capture degrees of freedom at the sequential level, and then consider the ensuing optimization problem.

A first attempt to the structural optimization of general synchronous circuits is the retiming/resynthesis strategy (considered, for example, in [18, 16, 17]). It essentially consists of identifying pipeline-like subcircuits, pushing the registers to their periphery, and then optimizing the resulting combinational subcircuit. The following very simple example shows the limitations of the method.

29th ACM/IEEE Design Automation Conference®

0738-100X/92 $3.00 © 1992 IEEE

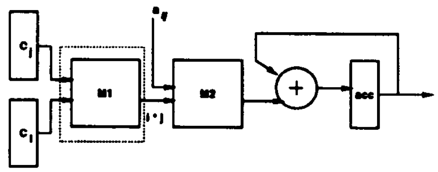

Figure 1: A circuit for a second-order moment computation.

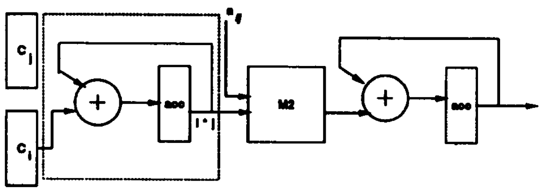

Figure 2: Optimization involving sequential information.

Example 2. Consider the circuit shown in Fig. (3). It can easily be verified that the inverter driving the variable y can be replaced by a simple interconnection, *i.e.* that the function $f(x) = x'$ can be replaced by $g(x) = x$. Since there are no pipeline-like subcircuits, no retiming operation is possible on the circuit, and consequently retiming would not remove the inverter. The inverter can be removed even though there are no *don't care* conditions associated to it in the combinational sense. To check this, it suffices to observe that any *don't care* condition on $f(x) = x'$ would result in the possibility of replacing the inverter with a constant 1 or 0, which is clearly incorrect. □

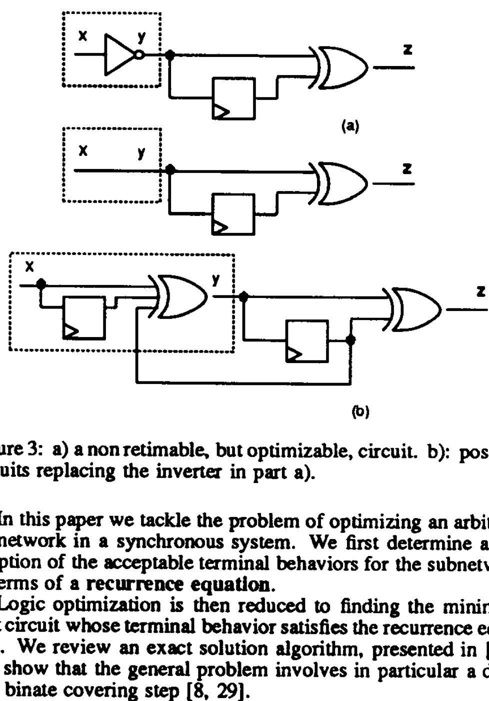

Figure 3: a) a non retimable, but optimizable, circuit. b): possible circuits replacing the inverter in part a).

In this paper we tackle the problem of optimizing an arbitrary subnetwork in a synchronous system. We first determine a description of the acceptable terminal behaviors for the subnetwork in terms of a **recurrence equation**.

Logic optimization is then reduced to finding the minimum cost circuit whose terminal behavior satisfies the recurrence equation. We review an exact solution algorithm, presented in [22], and show that the general problem involves in particular a difficult binate covering step [8, 29].

We then present a strategy for improving the efficiency of the optimization strategy by avoiding the binate covering step, so that conventional covering approaches can be considered. We conclude by presenting experimental results on synchronous logic benchmarks.

2 Synchronous Recurrence Equations.

We first introduce the conventional terminology associated to the manipulation of finite sequences (or **strings**) of Boolean values.

Let $\mathcal{B}$ denote the Boolean set $\{0,1\}$. A k-dimensional Boolean vector $\mathbf{x} = [x_1, \cdots, x_k]^T$ is an element of the set $\mathcal{B}^k$.

The set of all finite sequences over a finite set S (the *Kleene closure* of S) is conventionally denoted by the symbol S^* [21]. We thus denote by $(\mathcal{B}^k)^*$ the set of all finite sequences of k-dimensional Boolean vectors. An element of $(\mathcal{B}^k)^*$ is termed a **synchronous sequence** and denoted by $\mathbf{x}(\cdot)$. The n^{th} element of the sequence is denoted by $\mathbf{x}_n$.

The terminal behavior of a n_i-input, n_o-output synchronous circuit is described by the correspondence it establishes between input and output sequences, each pair representing a possible *execution trace* [27, 28] for the circuit. An execution trace is thus identified as an element of $(\mathcal{B}^{n_i+n_o})^*$.

In general, external specifications do not impose a unique correspondence between input and output sequences, but rather a *relation* between them, *i.e.* an arbitrary set of traces. Intuitively, this is due to: a) not all sequences are usually possible at the inputs of a synchronous circuit, and b) for a given input sequence, usually more output sequences are permitted.

Definition 2.1 *A* **trace set specification** *of an n_i-input, n_o-output synchronous circuit is a subset $\mathcal{T} \subseteq (\mathcal{B}^{n_i+n_o})^*$.*

A sequential synthesis problem could be formulated as follows:

Synchronous synthesis problem.

given	a trace set $\mathcal{T}$,
determine	an optimum circuit whose terminal behavior is contained in $\mathcal{T}$.

Given an arbitrary network, the extraction of the individual trace sets associated to its subnetworks is in general a complex task. For this reason, we first focus on **definite** (or feedback-free) networks. Any network can be decomposed into a definite portion, containing all the logic and delay elements, and a set of feedback interconnections. We first consider the problem of optimizing the definite portion, by assuming the feedback inputs and output to be ordinary primary inputs and outputs, respectively. In Sect. (3.3) we take feedback into account.

The terminal behavior of a definite network at time n is specified by a function $S(\mathbf{x}_n, \mathbf{x}_{n-1}, \ldots, \mathbf{x}_{n-d})$. [1] Usually, this specification is implicitly provided by the initial network.

We also assume that the external *don't care* conditions associated to the network can be described by an expression $DC(\mathbf{x}_n, \mathbf{x}_{n-1}, \ldots, \mathbf{x}_{n-d})$. Such *don't cares* represent the input sequences (of length up to d) that either do not occur or such that the network output is not observed [23].

The knowledge of S and DC identifies precisely the set of possible terminal behaviors for the network. A network realizing a function $F(\mathbf{x}_n, \ldots, \mathbf{x}_{n-d})$ meets such specifications if and only if $F = S$ for every sequence not in DC.

Another, equivalent, description of the set of terminal behaviors is in terms of the functions $F_{min} = S \cap DC'$ and $F_{max} = S \cup DC$. Specifications are met by F if

$$F_{min} \subseteq F \subseteq F_{max}$$

[1] Boolean functions can be seen as describing sets. Therefore they are referenced also as sets in this paper. Union and intersection denote sum and product, respectively. $\supseteq$ and $\subseteq$ denote containment.

The following examples show some contexts in which trace sets arise naturally in the optimization of synchronous circuits.

Example 3. In the circuit in Fig. (3a), we attempt to replace the input inverter by a simpler logic gate, producing the internal signal y. The replacement is possible as long as the global terminal behavior is unaffected. The desired terminal behavior for the network is described by $z_n = S(x_n, x_{n-1}) = x'_n \oplus x'_{n-1}$. The primary output z_n can also be expressed in terms of the internal signal y (to be re-synthesized) as $z_n = y_n \oplus y_{n-1}$. The signal y must therefore satisfy the constraint:

$$x'_n \oplus x'_{n-1} \subseteq y_n \oplus y_{n-1} \subseteq x'_n \oplus x'_{n-1}, \quad \forall n \geq 0$$

In this case, $n_i = n_o = 1$, and the above equation represents the constraint on the sequences of (x, y) pairs in the circuit.
For any given input sequence $x(\cdot)$, there exist more than one output sequence $y(\cdot)$ that satisfies the equation. Two possible solutions are

$$y_{-1} = x_{-1}; \quad y_n = x_n \;\; \forall n \geq 0; \qquad y_{-1} = 0, \quad y_n = x_n \oplus x_{n-1} \oplus y_{n-1} \;\; \forall n \geq 0.$$

(The second solution is obtained simply by adding y_{n-1} to both terms of the equation).
The solutions correspond non equivalent circuits replacing the original inverter, shown in Fig. (3b). The assignments of y_{-1} correspond to the assignment of the *initial conditions* for the subcircuit. Although in this case the original circuit is combinational, the second solution is not, and the new network contains a feedback interconnection. □

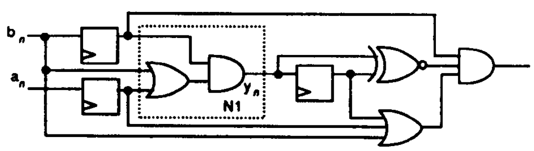

Figure 4: Circuit for the optimization problem of Example (4)

Example 4. As a more complex example, consider the optimization of the subnetwork N1 in Fig. (4). The desired terminal behavior of the entire network can be described by

$$S = b_{n-2} b_{n-1}(a_{n-1} + b_n)$$

Its output is expressed in terms of the internal signal y by:

$$b_{n-1}[y_n \overline{\oplus} y_{n-1}](b_n + a_{n-1} + y_{n-1})$$

Therefore, for every input sequence, y must satisfy

$$b_{n-1}[y_n \overline{\oplus} y_{n-1}](b_n + a_{n-1} + y_{n-1}) = S$$

which represents the recurrence equation to be satisfied by any subnetwork generating the signal y.
Suppose that a *don't care* information (say, $DC = a_n(a_{n-1} + b_n)$) is added. The above equality then would have to be satisfied only for those sequences not in DC. Consequently, y must satisfy only the recurrence:

$$S\,DC' \subseteq b_{n-1}[y_n \overline{\oplus} y_{n-1}](b_n + a_{n-1} + y_{n-1}) \subseteq S + DC$$

□

In all the above examples trace sets are most naturally represented implicitly as solutions to a *recurrence equation*, involving the elements $\mathbf{y}_n, \ldots, \mathbf{y}_{n-d}$ of the output sequences of the circuit to be synthesized. We thus introduce the following definition:

Definition 2.2 *A* **Synchronous Recurrence Equation** *(SRE) is a Boolean equation of type*

$$F_{min} \subseteq F(\mathbf{x}_n, \cdots, \mathbf{x}_{n-2d}, \mathbf{y}_n, \cdots, \mathbf{y}_{n-d}) \subseteq F_{max}; \quad \forall n \geq 0.$$

A **feasible solution** *of the SRE is a function*

$$\mathbf{f}(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}, \mathbf{y}_{n-1}, \cdots, \mathbf{y}_{n-d})$$

and an initial value specification

$$\mathbf{y}_{-d} = \mathbf{g}_{-d}(\mathbf{x}_{-d}, \cdots, \mathbf{x}_{-2d})$$
$$\vdots$$
$$\mathbf{y}_{-1} = \mathbf{g}_{-1}(\mathbf{x}_{-1}, \cdots, \mathbf{x}_{-d})$$

such that if

$$\mathbf{y}_n = \mathbf{f}(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}, \mathbf{y}_{n-1}, \cdots, \mathbf{y}_{n-d}) \;\; \forall n \geq 0$$

then Eq. 2.2 holds true.

3 Solving the SRE.

We consider in this paper terminal specifications for a synchronous circuit provided in form of a SRE. Each solution $\mathbf{f}$ to the SRE corresponds to a possible realization of such specifications, with an associated cost. The task of logic synthesis is in this case to determine the minimum cost (typically, minimum-hardware) synchronous circuit whose terminal behavior satisfies the SRE:

Synthesis from SRE.

given	an SRE,
determine	its minimum cost feasible solution (if one exist).

A synchronous network realizing a function as in Eq. 2.2 may in general contain feedback interconnections, as $\mathbf{y}_n$ is expressed in terms of the past values $\mathbf{y}_{n-1}, \cdots, \mathbf{y}_{n-d}$ (one such example is shown in Fig. (3b)). We are not interested in this type of solutions, as they alter the network topology during optimization.
We therefore focus our attention on simpler solutions, in the form $\mathbf{f}(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$ only. These solutions, yielding feedback-free (or **definite**) realizations, are hereafter termed **definite**.

3.1 Definite solutions.

We focus now on the optimization of a single-output function f, the extension to the multiple-output case being straightforward. An exact solution procedure for two-level expressions of f was outlined in [22]. We define a **cube** $c(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$ on the variables of $\mathbf{x}_0, \cdots, \mathbf{x}_{-d}$ as the product of some of such variables, in either true or complemented form (in practice, we allow also outputs of other internal gates to appear as factors of a cube).
We seek expressions of f as a sum of cubes:

$$f = \sum_{k=1}^{N} c_k.$$

A cube c is an **implicant** if there exists a feasible solution f containing c. An implicant c is a **prime** if there exists a feasible solution f for which c is prime, *i.e.* for which no implicant c' of f strictly contains c. A procedure is outlined in [22] to determine the prime implicants of a SRE.

Example 5. The primes for Example (4) are shown in Table (1). □

Once the set S of primes has been built, Petrick's method is used to construct the subsequent covering problem [26, 7, 8] as follows. The general solution is written as

$$f = \sum_{r=1}^{|S|} \alpha_r c_r$$

Primes
$c_1 = b'_{n-1}$
$c_2 = a_{n-1}b_{n-1}$
$c_3 = b'_n a_{n-1}$
$c_4 = b_n b_{n-1}$

Table 1: List of primes for the problem of Example (4).

where the parameter variable α_r is 1 if c_r is present in the solution, and $\alpha_r = 0$ otherwise.

This expression of f replaces y in the SRE, to obtain a new equation in terms of the input variables $\mathbf{x}_i$ and the parameters α.

Example 6. Table (1) contains the primes for Example (4). Corresponding to the assignment $b_n = b_{n-1} = 1; a_{n-1} = a_{n-2} = b_{n-2} = 0$ the SRE reduces to

$$0 \subseteq y_n \overline{\oplus} y_{n-1} \subseteq 0$$

On the other hand, corresponding to that assignment, $y_n = \alpha_4$ and $y_{n-1} = \alpha_1$. It must therefore be $\alpha_1 \oplus \alpha_4 = 1$.
By repeating the same process over all assignments, and by forming the product of all the resulting constraints on the α_i, the global constraint equation (already in conjunctive normal form) follows:

$$SRE_\alpha = (\alpha_1 + \alpha_2 + \alpha_3)(\alpha'_1 + \alpha'_3)(\alpha'_1 + \alpha'_2)(\alpha_1 + \alpha_4)(\alpha'_1 + \alpha'_4) = 1.$$

In particular, the last two products correspond to the factors of $\alpha_1 \oplus \alpha_4$.
The minimum-cost solution to $SRE_\alpha = 1$ is represented by $\alpha_1 = 1, \alpha_2 = \alpha_3 = \alpha_4 = 0$, corresponding to $f = b'_{n-1}$. □

The synthesis problem is thus eventually reduced to that of finding the minimum cost assignment to the parameters α_r such that the new equation holds for all assignments of the variables x_i. This problem is known in the literature as **Minimum Cost Satisfiability** or **Binate Covering** problem.

Its binate nature comes from the possibility for some of the parameter variables α_i to appear in both forms, true and complemented, in the conjunctive form of SRE_α. For instance, the variable α_1 appears in both forms in the SRE_α of Example (6).

3.2 Unate Covering Problems

The potentially exponential number of primes and the binate nature of the covering step represent the difficulties associated with the optimization problem.

Binate covering may arise simply because of the nature of the function F in the SRE. In particular, if F contains $y_n, \cdots, y_{n-d}$ in both true and complemented forms, then the coefficients α_i will generally appear in SRE_α with mixed polarity.

The binate nature of the covering problem reflects an intrinsic difficulty in the covering step. In the unate case, the effect of adding / removing a prime to a partial solution is always predictable: we increase / decrease the cover given by that partial solution. This is not necessarily true in the present case. Since F is of mixed polarity w.r.t. $y_n, \cdots, y_{n-d}$, the effect on F of adding a cube to y becomes unpredictable.

It is thus desirable to remove the mixed-polarity dependency of F on y. Such a dependency occurs only if there are paths from the gate under optimization to the primary outputs with different parity of inversions.

Definition 3.1 *A function* $F(\mathbf{y}_n, \ldots, \mathbf{y}_{n-d})$ *is termed* **positive (negative) unate** *in* $\mathbf{y}$ *if it is positive (negative) unate in each of* $\mathbf{y}_n, \ldots, \mathbf{y}_{n-d}$.

Definition 3.2 *A network is termed* **unate** *w.r.t. a gate* **g** *if all reconvergent paths from* **g** *have the same parity of inversions. A network is* **internally unate** *if it is unate w.r.t. each of its gates.*

The output of a network unate w.r.t. a gate **g** is a unate function of the gate output. Any network can be made internally unate by duplicating the gates with different parity of inversions in their fanout paths. The new network is at most twice the size the original one. In practice, the increase is generally much smaller.

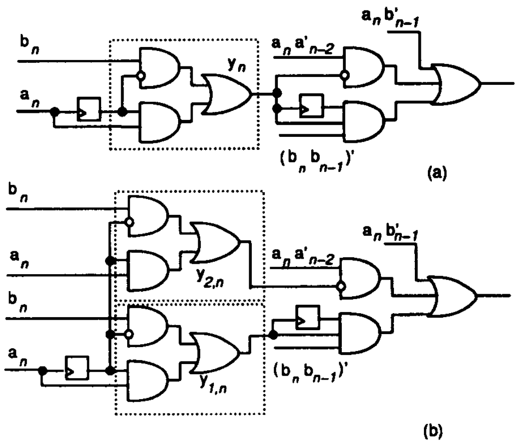

Figure 5: Unate transformation of a network.

Example 7. In the circuit of Fig. (5a), we seek the optimization of the gate with output y. The network realizes the function

$$S = a_n b'_{n-1} + a_n b'_n (a_{n-1}(a_{n-2} + b_{n-1}) + a'_{n-1} a'_{n-2})$$

We assume no external *don't care* conditions. Consequently, $F_{min} = F_{max} = S$. The output is expressed in terms of the y by

$$F = a_n b'_{n-1} + a_n a'_{n-2} y'_n + (b_n b_{n-1})' y_n y_{n-1}$$

Fig. (5b) shows the unate version of the example in Fig. (5a). The gate to be optimized has been duplicated, the two new outputs being y_1, y_2. The output is now a unate function of each of y_1, y_2:

$$F = a_n a'_{n-2} y'_{2,n} + (b_n b_{n-1})' y_{1,n} y_{1,n-1}$$

□

Primes
$c_1 = a_n$
$c_2 = b_n$
$c_3 = a_n a_{n-1}$
$c_4 = b_n b_{n-1}$
$c_5 = a_n b'_{n-1}$
$c_6 = b_n a'_{n-1}$

Table 2: List of primes for the problem of Example (7).

The following example shows that the unateness of F alone, however, is not a sufficient condition to insure unateness of covering problems.

Example 8. Table (2) contains a list of primes of y_1. The prime b_n, however, can never appear in the same cover with, for example, $a_n a_{n-1}$. This can be verified as follows. Consider, for instance, a cover of y containing $a_n a_{n-1} + b_n$. The product $y_n y_{n-1}$ would then contain the cross-product $b_n a_{n-1} a_{n-2}$. Corresponding to the assignment $b_n = a_{n-1} = a_{n-2} = 1; a_n = b_{n-1} = 0$ we have $F = 1$ and $F_{max} = 0$, thereby violating the constraint $F \subseteq F_{max}$. □

In the case of unate covering problems, two partial covers can always be merged to obtain a larger valid cover. In Example (8) instead, the appearance of the cross-product $b_n a_{n-1} a_{n-2}$ invalidates the union of any two covers containing $a_n a_{n-1}$ and b_n, respectively, by violating $F \subseteq F_{max}$.

In order to construct unate covering subproblems, a more accurate understanding of the effect of adding / expanding an implicant to a partial cover is therefore necessary.

Assume, for simplicity, that F is positive unate in **y**, and consider the effect on F of elementary operations, such as the addition / expansion of an implicant to **y**. Due to the positive unateness of F in **y**, corresponding to each assignment of $\mathbf{x}_n, \ldots, \mathbf{x}_{n-d}$, an elementary operation can at most change $F = 0$ to $F = 1$, but not viceversa. Each elementary operation therefore preserves the inequality $F_{min} \subseteq F$, and only the second inequality $F \subseteq F_{max}$ needs be checked. In Example (8), precisely this constraint is violated by the cover $a_n a_{n-1} + b_n$. This leads to the following definition:

Definition 3.3 *(Maximal set of primes.) A set of implicants is said to be* **maximal** *if no implicant can be expanded / added to it without violating $F \subseteq F_{max}$.*

As Example (9) implicitly points out, there may be more than one maximal set of primes.

Example 9.
For the circuit shown in Fig. (5), the following are maximal sets of primes for the variable y_1:

$$S_1 = \{a_n b_n, a_n a_{n-1}, b_n b_{n-1}, a_n b'_{n-1}, b_n a'_{n-1}\};$$

$$S_2 = \{b_n\}.$$

$$S_3 = \{a_n, b_n a'_{n-1}\};$$

□

Once a maximal set of primes has been built, the removal of any primes can only change $F = 1$ to $F = 0$ for some assignments of $\mathbf{x}_n, \ldots, \mathbf{x}_{n-d}$. Therefore, removal of any primes can never result in a violation of $F \subseteq F_{max}$. The covering step can thus be formulated as

minimize the number of implicants/literals of y;
constraint: $F_{min} \subseteq F$.

To achieve exact minimization, all possible maximal sets of primes should be generated and the subsequent covering step solved. This is obviously inefficient and costly in terms of CPU time. A useful approximation is instead that of building only one maximal set of primes upon the original set. For example, for the network of Fig. (5) only S_1 or S_3 would be derived.

Once the internally unate network is optimized, it can be folded back into an arbitrary, internally binate, network.

Example 10. Beginning from the set S_1 of Example (9), the optimum two-level realization of y_1 is $a_n a_{n-1} + b_n b_{n-1}$. It could then be shown that an optimal realization of y_2 is b'_n. The network after the above transformations is shown in Fig. (6). □

3.3 Adding Feedback.

In the optimization process followed so far, we treated the feedback inputs and outputs as ordinary primary inputs and outputs,

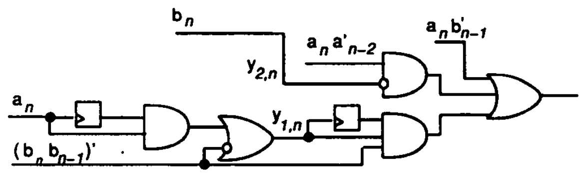

Figure 6: The optimized version of the network.

respectively. In practice, at any point in time the value at a feedback input is fixed by the network, and we expect that not all feedback sequences will be possible. Moreover, the feedback outputs are not perfectly observable. We can model both effects by means of suitable *don't care* conditions, as shown by the following example.

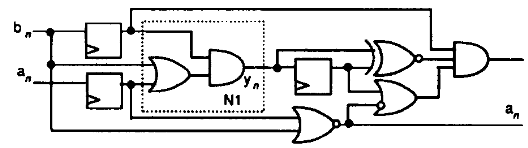

Figure 7: *Don't care* conditions for a circuit with feedback.

Example 11. Consider the network of Fig. (7). It represents essentially the same network of Fig. (4). In this case, however, a_n is not an external signal, but rather a feedback input. In particular, $a_n = (a_{n-1} + b_n)'$. We can therefore add an external controllability *don't care* set $DC = a_n \oplus (a_{n-1} + b_n)'$ to the specifications.
The input a_n is also not observed at any time in the future if $b_{n+1} = 1$. We can therefore associate an observability *don't care* set b_{n+1} to the feedback output. □

Unlike the combinational case, however, the controllability and observability *don't care* conditions derived from the feedback interconnection cannot generally be used simultaneously in the optimization of the network. Indeed, using the observability *don't care* set implies changing the feedback function, and possibly invalidating the corresponding controllability *don't care* set.

3.4 Experimental Results

The algorithms outlined in this paper have been written in C and tested on standard synchronous logic benchmarks. The results of applying the exact method of Sect. (3.1) are shown in Table 3.4. They were obtained on a DEC 5000 workstation. In particular, the first four columns refer to the initial benchmark statistics, in terms of inputs, outputs, literal, and register counts, respectively. Column *optl* reports the final number of literals and registers obtained, while *cpu* shows the CPU time in seconds.

4 Summary and Conclusions.

In this paper we outlined a model for the optimization of synchronous circuits across register boundaries. We use the concept of **sets of execution traces**, rather than state diagrams, to specify the desired terminal behavior (and its associated degrees of freedom) of a synchronous circuit.

For synchronous logic optimization problems, **Synchronous Recurrence Equations** represent an efficient way of representing trace sets. We can thus cast the synthesis problem of a syn-

Circuit	inputs	outputs	lits	regs	optl	cpu
s208	11	2	166	8	108	3
s298	3	6	244	14	155	14
s344	9	11	269	15	186	25
s420	19	2	336	16	251	258
s444	3	6	352	21	202	142
s641	35	24	539	19	241	302

Table 3: Experimental results for some logic optimization benchmarks.

chronous circuit as that of finding the minimum-cost solution to such equations.

An exact two-step solution algorithm has been proposed. The first step transforms the synchronous problem into a combinational one, which we have shown to differ from those previously considered in the literature. An exact algorithm for the latter problem is then presented.

We have shown that the solution of this problem involves a difficult **binate covering** step, for which efficient algorithms are under investigation [29]. To overcome this difficulty, we have shown that by making the network **internally unate** it is possible to reduce the covering step to a much simpler unate one.

Experimental results show that this approach is viable for the hierarchical optimization of synchronous circuits, with a solution space not otherwise reachable.

5 Acknowledgements.

This research was sponsored by DEC, jointly with NSF, under a PYI Award and by NSF/DARPA, under contract # MIP 8719546 and its renewal.

References

[1] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang, "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, No. 6, pp. 1062-1081, November 1987.

[2] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multilevel Logic Minimization Using Implicit Don't Cares", *IEEE Transactions on CAD/ICAS*, vol. CAD-7, No. 6, pp. 723-739, June 1988.

[3] S. Muroga, Y.Kambayashi, H.Lai and J.Culliney, "The Transduction Method - Design of Logic Networks Based on Permissible Functions", *IEEE Trans. Comp.*, vol. 38, No. 10, pp. 1404-1424, 1989.

[4] E. J. McCluskey, " Minimization of Boolean Functions," *Bell Syst. Tech. Jour.*, vol. 35, pp. 1417-1444, 1956.

[5] W. V. Quine, " The Problem of Simplifying Truth Functions ", *Am. Math. Monthly*, vol. 59, pp. 521-531, 1952.

[6] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Boston, Kluwer Academic Publishers, 1984.

[7] R. K. Brayton and F. Somenzi, " Boolean Relations and the Incomplete Specification of Logic Networks", *IFIP VLSI 89 Int. Conference*, pp. 231-140, Munich, 1989.

[8] R. K. Brayton and F. Somenzi, " An Exact Minimizer for Boolean Relations", *Proc. ICCAD 1989*, pp. 316-319, S. Clara, Nov. 1989.

[9] S.Devadas, T.Ma, A. Newton, and A.Sangiovanni-Vincentelli, "A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines", *IEEE Transactions on CAD/ICAS*, Vol. CAD-8 No. 10, pp. 1100-1109 October 1989.

[10] S. Devadas and A. R. Newton, " Decomposition and Factorization of Sequential Finite State Machines", *IEEE Trans. on CAD*, vol. 8, pp. 1206-1217, 1989.

[11] G. Saucier, M. Crastes de Paulet and P. Sicard, "ASYL: A Rule-Based System for Controller Synthesis", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, pp. 1088-1097 November 1987.

[12] J. Hartmanis and H. Stearns, *Algebraic Structure Theory of Sequential Machines*, Englewood Cliffs, N.J., Prentice-Hall, 1966.

[13] Z. Kohavi, *Switching and Finite Automata Theory*, 2^{nd} ed., New York, Englewood Cliffs, N.J., Prentice-Hall [1966]) McGraw-Hill, 1978.

[14] K. T. Cheng and V. D. Agrawal, "State Assignment for Initializable Synthesis", *Proc. ICCAD 1989*, S. Clara, Nov. 1989.

[15] G. De Micheli, " Synchronous Logic Synthesis: Algorithms for Cycle-Time Optimization", *IEEE Trans. on CAD*, vol. 10, pp. 63-73, 1991.

[16] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, " Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques", *IEEE Trans. on CAD*, vol. 10, pp. 74-84, 1991.

[17] S. Dey, F. Brglez, and G. Kedem, " Partitioning Sequential Circuits for Logic Optimization", *Proc. 3^{rd} Int'l Workshop on Logic Synthesis*, Research Triangle Park, 1991.

[18] C. E. Leiserson, F. M. Rose, and J. B. Saxe, " Optimizing Synchronous Circuitry by Retiming", in R. Bryant, ed., *Proc. 3^{rd} CALTECH Conference on Large Scale Integration*, Computer Science Press, Rockville, 1983.

[19] W. Wolf, A. Takach, and T. C. Lee, " Architectural Optimization Methods for Control-Dominated Machines", in Camposano and Wolf (editors): *High-Level VLSI Synthesis*, pp. 231-254, Kluwer, 1991.

[20] D. Ku, D. Filo and G. De Micheli, " Control Optimization Based on Resynchronization of Operations", *Proc. 1991 DAC Conf.*, S. Francisco, June 1991.

[21] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.

[22] M. Damiani and G. De Micheli, " Synthesis and Optimization of Synchronous Logic Circuits from Recurrence Equations", *Proc. EDAC 1992*, Bruxelles, March 1992.

[23] M. Damiani and G. De Micheli, " *Don't care* conditions in Combinational and Synchronous Logic Networks", To appear on *IEEE Trans. on CAD*.

[24] M. Damiani and G. De Micheli, "The Role of *don't care* conditions in Synchronous Logic Optimization", in *Proceedings of Synthesis and Simulation Meeting and International Interchange (SASIMI)*, pp. 55-62, Tokio, 1990.

[25] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1988.

[26] S. R. Petrick, " A Direct Determination of the Irredundant Forms of a Boolean Function from a set of Prime Implicants, " A. F. Cambridge Res. Center Report AFCRC-TR-56-110, Bedford, Mass., 1956.

[27] D. L. Dill, *Trace Theory for Automatic Verification of Speed Independent Circuits*, MIT Press, Cambridge, 1988.

[28] M. Rem, J. L. A. VanDeSnepscheut and J.T. Udding, "Trace Theory and the Definition of Hierarchical Components", in R. Bryant, ed., *Proc. 3^{rd} CALTECH Conference on Large Scale Integration*, Computer Science Press, Rockville, 1983.

[29] M. Pipponzi and F. Somenzi, " An Iterative Algorithm for the Binate Covering Problem", *Proc. EDAC 1989*, pp. 208-211, Glasgow, March 1989.