Synthesis and Optimization of Synchronous Logic Circuits from Recurrence Equations.

Maurizio Damiani Giovanni De Micheli

Center for Integrated Systems
Stanford University

Abstract

In this paper we present a general solution framework for optimizing synchronous networks across register boundaries. We formulate the problem as that of finding minimum-cost solutions to **Synchronous Recurrence Equations.** *We propose an algorithm for the solution of such equations that relies on their transformation into a new combinational logic optimization problem. An exact solution algorithm for this problem is presented, and experimental results on synchronous benchmark circuits demonstrate the feasibility of the approach.*

1 Introduction.

Synthesis and optimization problems for combinational and synchronous logic circuits are currently the object of intense investigation. Synthesis methods for combinational networks have achieved, over the past few years, a significant level of maturity. In particular, several theoretical aspects of two-level [14, 16] and hierarchical multiple-level [1, 2, 3, 20, 22] optimization are well understood, and effective tools exist that take advantage of those theory. Such methods work directly on structural representations of the logic network, and rely upon the ability to manipulate sets (such as *don't cares* [2] or *equivalence classes* [22]) by means of Boolean expressions.

By contrast, the optimization of synchronous circuits has so far relied on procedures based on (possibly iterative) manipulations of their *behavioral models*, typically in terms of *state transition graphs* [9, 10, 4, 5, 11, 6]. The major 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 and performance, during the optimization process. Consequently, recent research efforts [7, 8, 21] are focusing on methods that, similarly to the combinational case, can optimize synchronous circuits from *structural* models, *i.e.* netlists, according to given area/performance metrics.

In the combinational case, the basic optimization steps consist in iteratively extracting subnetworks to be optimized, identifying the degrees of freedom in their input/output specification, and then resorting to known algorithms for their optimization. In the case of single-output subnetworks, such degrees of freedom are fully represented by a *don't care* set associated to their function, due to the embedding of the function in a larger network [2]. For multiple-output subnetworks, it has been shown that the input/output specifications are best described by *Boolean relations* [20] (A Boolean relation associates a set of possible output assignments to each input assignment).

The structural model generally adopted for synchronous circuits is the **synchronous Boolean network** [7]. A synchronous Boolean network is described by a **network graph**, whose vertices correspond to logic gates, and directed edges to interconnections. Flip-flops are modeled by **unit delay elements**. The **retiming/resynthesis** operation [19] has been proposed for the optimization of such networks [19, 8, 21]. It essentially consist of identifying pipeline-like subcircuits, pushing all the delay elements to their periphery, and then optimizing the remaining combinational portion with combinational optimization techniques This approach does not fully capture the optimization space for synchronous networks, as shown by the following simple example.

Example 1. Consider the circuit shown in Fig. (1). 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. It is also interesting to observe that the inverter can be replaced even though there are no *don't care* conditions associated to it. To check this, it suffices to observe that any *don't care* condition on $f(x)$ would result in the possibility of replacing the inverter with a constant 1 or 0, which is clearly incorrect. □

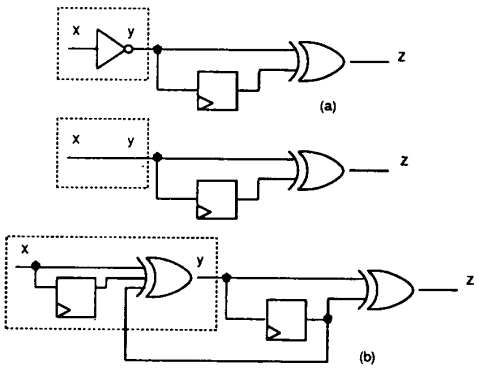

Figure 1: a) a non retimable, but optimizable, circuit. b): possible circuits replacing the inverter in part a).

In order to capture the degrees of freedom for a subnetwork embedded in a synchronous system, it is necessary to be able to fully describe the terminal specifications imposed on that subnetwork. In the synchronous case, the most general terminal specifications are represented by the set of its possible **execution traces** [17, 18]. A trace is defined as a pair of input/output *sequences*. For this reason, we consider **trace sets** as specifications for a synchronous subnetwork. In this paper, trace sets are implicitly described as solutions to **Synchronous Recurrence Equations.** In the next section we show that this type of description arises naturally from the optimization framework considered here.

The task of logic optimization is that of finding the minimum

0-8186-2645-3/92 $3.00 © 1992 IEEE

cost circuit whose input/output behavior satisfies the recurrence equation. We present an exact two-step solution algorithm for this problem. The first step consists of reducing the synchronous logic optimization problem to a combinational logic optimization one. We show that the reduced problem is more general than those so far considered in the literature [14, 16, 22]. We conclude the paper by presenting an exact solution algorithm for the reduced problem and some experimental results on synchronous benchmark circuits.

2 Synchronous Recurrence Equations.

2.1 Terminology

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^* [17]. 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 input / output functionality 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* [17, 18] for the synchronous circuit.

In general, external specifications do not impose a unique input/output correspondence, but rather a *relation* between input and output sequences, *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.

Trace sets represent the most general type of terminal specifications for synchronous circuits. The following examples show some contexts in which they arise naturally as specifications for a synchronous (or even combinational) circuit.

Example 2. In the circuit in Fig. (1a), we seek to replace the input inverter by a simpler logic gate, generating the intermediate signal y. The replacement is possible as long as the input/output behavior of the whole network is unaffected. The desired input/output behavior for the network is $z_n = x'_n \oplus x'_{n-1}$. The primary output z can 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:

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

The above equation represents the constraint on the execution traces by the circuit replacing the inverter. It is worth remarking that for any given input sequence $x(\cdot)$, there exist more than one output sequence $y(\cdot)$ that satisfy the equation. Two possible solutions are

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

In particular, the second solution is obtained by adding y_{n-1} to both terms of the equation.
Such solutions correspond to different circuits replacing the original inverter, shown in Fig. (1b). The assignments of y_{-1} correspond to the assignment of the *initial conditions* for the subcircuit. Note that, although in this case the original circuit is combinational, the second solution is not, and contains a feedback interconnection. □

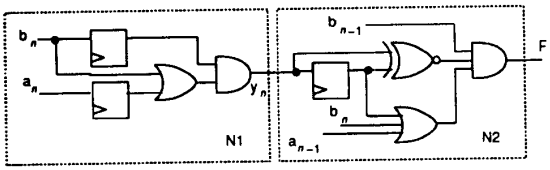

Figure 2: Circuit for Example (3)

Example 3. As a more complex example, consider the optimization of the subnetwork N1 in the cascade interconnection of Fig. (2). The desired input/output behavior of the entire network can be described by

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

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

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

Therefore, for every input sequence, y must satisfy

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

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

or, equivalently,

$$b_{n-2}b_{n-1}(a_{n-1} + b_n)\overline{\oplus}$$

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

which represents the recurrence equation to be satisfied by any subnetwork generating the signal y. □

Example 4. Fig. (3) shows the flow graph for a simple sequence of arithmetic operations. A control unit is to be designed that receives an input $cnta$, signaling the firing of OP_0, and activates the operations OP_1 and OP_2 by means of $cntb$ and $cntc$, according to some timing constraints. In this example, OP_1 must be activated either 1 or 2 clock cycles after the activation of OP_0, while OP_2 must be activated between 2 and 3 clock cycles after OP_1. Such constraints can be expressed as implications between the control signals $cnta(\cdot), cntb(\cdot), cntc(\cdot)$ at different time points. In particular, $cnta_{n-2} = 1$ implies that OP_1 must be activated either at time $n-1$ or at time n. This is expressed by

$$cnta_{n-2} \subseteq cntb_{n-1} + cntb_n \quad \forall n \geq 0$$

or, equivalently, by

$$cnta'_{n-2} + cntb_{n-1} + cntb_n = 1 \ \ \forall n \geq 0\,.$$

The second implication is that if OP_1 is activated at time n, then either $cnta_{n-1} = 1$ or $cnta_{n-2} = 1$. This is described by the clause

$$cnta_{n-1} + cnta_{n-2} + cntb'_n = 1.$$

Similarly, the activation of OP_2 is described by the constraints:

$$cntb'_{n-3} + cntc_{n-1} + cntc_n = 1$$

$$cntc'_n + cntb_{n-3} + cntb_{n-2} = 1$$

All of the above constraints must be satisfied simultaneously, and can be cast in a single product to form a recurrence equation to be satisfied by the signals $cnta(\cdot), cntb(\cdot)$. □

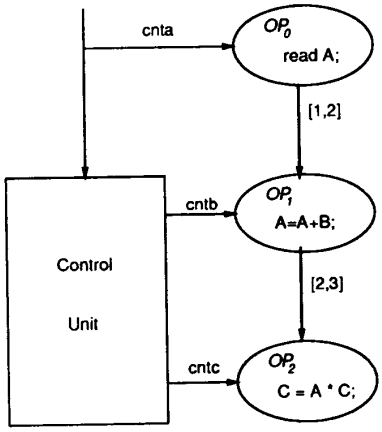

Figure 3: High-level flow graph and control unit for a sequence of operations.

In these examples trace sets are described as solutions to a *recurrence equation*, involving the elements of the input

and output sequences of the circuit to be synthesized. In Examples (2) and (3), the recurrence equation is generated by imposing the equivalence of the input/output behavior of the original and modified networks, whereas for the synthesis problem of Example (4) it is derived directly from high-level specifications. We thus introduce the following definition:

Definition 1. *We call* **Synchronous Recurrence Equation** *(SRE) a Boolean equation of type*

$$\mathcal{R}(\mathbf{x}_n, \ldots, \mathbf{x}_{n-2d}, \mathbf{y}_n, \ldots, \mathbf{y}_{n-d}) = 1; \ \forall n \geq 0.$$

The Boolean function $\mathcal{R}$ is the **characteristic function** *of the equation. The (nonnegative) integer d is termed the* **memory depth** *of the equation. We call a* **feasible solution** *of the SRE 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. 2.1 holds true. □

We assume in this paper that the input/output specifications for a synchronous circuit are provided in the form of an 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 input/output behavior satisfies the SRE:

Synchronous synthesis problem.

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

A synchronous network realizing a function as in Eq. 2.1 may in general contain feedback interconnections, as $\mathbf{y}_n$ is expressed in terms of the past values $\mathbf{y}_{n-1}, \cdots, \mathbf{y}_{n-d}$. In this paper we focus our attention on simpler solutions, in the form $\mathbf{f}(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$ only, and defer the description of the general synthesis procedure to a later paper, for reasons of space. The solutions considered here, yielding feedback-free (or **definite**) realizations, are hereafter termed **definite**. Definite solutions do not need initial value specifications.

3 Finding definite solutions to the SRE.

Our solution procedure is essentially divided in two steps. The first step consists of transforming the synchronous synthesis problem into a combinational one, by providing a characterization of the feasible solutions to an SRE.

We recall that Eq. 2.1 represents a **functional equation**, *i.e.* an equation in which the unknown is the function $\mathbf{f}(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$. In the Boolean domain, $\mathbf{f}$ is completely described by its truth table; we can thus regard the truth table entries of the function $\mathbf{f}$ as the actual unknowns of the problem. The first step will thus consist of determining a representation of the truth tables corresponding to feasible solutions. The second step consists in the search procedure for optimum solutions. We focus in particular on minimum two-level representations of $\mathbf{f}$.

3.1 Representing feasible solutions

For the sake of simplicity, in this paper we limit our attention to the synthesis of a single-output function f, the generalization to the multiple-output case being straightforward. The support of f is formed by the $n_i \times d$ variables representing the components of the vectors $\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}$. Any such function can be represented by its truth table, of $2^{n_i \times d}$ entries. These entries are here denoted by $f_j; j = 0, \cdots, 2^{n_i \times d} - 1$.

A function f is completely specified once all f_j's have been assigned a value. At the beginning of the solution process, none of the f_j are known, and there are in general several possible assignments, corresponding to feasible solutions of different cost.

Example 5. For the problem of Example (1), we seek a function $f(x_n, x_{n-1})$ of minimum cost that can replace the inverter. The function is entirely described by its truth table, represented in Table 1. The entries f_0, f_1, f_2, f_3 represent the unknowns of the problem. Definite feasible solutions are represented by $f_0 = 1, f_1 = 1, f_2 = 0, f_3 = 0$ (corresponding to the original inverter) and by $f_0 = 0, f_1 = 0, f_2 = 1, f_3 = 1$ (corresponding to the simple interconnection). □

For each assignment of $\mathbf{x}_n, \cdots, \mathbf{x}_{n-2d}$, Eq. 1 specifies a constraint on the possible assignments to $y_n, \cdots, y_{n-d}$. Such constraints can be expressed by means of a **relation table** associated to the characteristic function $\mathcal{R}$. The left-hand side of the table represents the assignments of the inputs $\mathbf{x}_n, \cdots, \mathbf{x}_{n-2d}$, while its right-hand side represents the corresponding assignments to $y_n, \cdots, y_{n-d}$ that result in $\mathcal{R} = 1$.

$x(n)$	$x(n-1)$	f
0	0	f_0
0	1	f_1
1	0	f_2
1	1	f_3

Table 1: Symbolic tabular representation of an unknown function $f(x_n, x_{n-1})$.

Example 6. For the problem of Example (1),

$$\mathcal{R} = (x'_n \dotplus x'_{n-1}) \overline{\oplus} (y_n \oplus y_{n-1}).$$

Corresponding to the assignment, say, $(x_n, x_{n-1}, x_{n-2}) = (0, 1, 0)$, the equation $\mathcal{R} = 1$ reduces to the constraint $y_n \oplus y_{n-1} = 1$. Table 2 contains the relation table for $\mathcal{R}$, and in particular the second column shows the assignments of y_n, y_{n-1} that satisfy $\mathcal{R} = 1$, corresponding to each assignment of x_n, x_{n-1}, x_{n-2}. The relation table for the problem of Example (3) is shown in Table 3. □

x_n	x_{n-1}	x_{n-2}	$y_n\ y_{n-1}$	$y_n\ y_{n-1}$
0	0	–	00, 11	$(f'_0 + f_0)(f_0 + f'_0) = 1$
0	1	–	01, 10	$(f'_1 + f'_3)(f_1 + f_3) = 1$
1	0	–	01, 10	$(f'_2 + f'_1)(f_2 + f_1) = 1$
1	1	–	00, 11	$(f'_3 + f_3)(f_3 + f'_3) = 1$

Table 2: Relation table for the inverter optimization problem. The second column shows the possible assignments to y_n, y_{n-1} corresponding to each input sequence; the third one expresses those assignments in terms of the entries f_j.

a_n	b_n	a_{n-1}	b_{n-1}	a_{n-2}	b_{n-2}	$y_n\ y_{n-1}$
-	-	-	0	-	-	–
-	0	0	1	-	-	0–, –0
-	-	1	1	-	0	01, 10
-	1	-	1	-	0	01, 10
-	-	1	1	-	1	00, 11
-	1	-	1	-	1	00, 11

Table 3: Relation table for the problem of Example (3). Dashes represent *don't care* conditions.

We recall at this point that we are seeking solutions in the form $y_n = f(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$. Corresponding to each entry of the

relation table, we can re-express the right-hand side constraints on $y_n, \cdots, y_{n-d}$ as constraints on the f_j's, as shown by the following example.

Example 7. For the relation table of Table 2, corresponding to the assignment $(x_n, x_{n-1}, x_{n-2}) = (0,0,1)$, the possible assignments for (y_n, y_{n-1}) are either $(0,0)$ or $(1,1)$, *i.e.* it must be

$$(y_n + y'_{n-1})(y'_n + y_{n-1}) = 1.$$

Since we assume $y_n = f(x_n, x_{n-1})$ and $y_{n-1} = f(x_{n-1}, x_{n-2})$, we have $y_n = f(0,0) = f_0$ and $y_{n-1} = f(0,1) = f_1$. Therefore, the possible assignments for y_{n-1}, y_n are also described by

$$(f_0 + f'_1)(f'_0 + f_1) = 1.$$

The same process is repeated for all rows of the relation table. The resulting constraints on the entry variables f_j are described in column 3 of Table 2. □

A function f represents a feasible solution to an SRE if and only if all the constraints appearing on the right-hand side of the relation table hold true. It is thus possible to represent such constraints by means of their conjunction, *i.e.* by a single equation of type

$$\mathcal{K}(f_j; j = 0, \cdots, 2^{n_i \times d} - 1) = 1.$$

Example 8. In the problem of Example (3), by considering a solution in the form $f(b_n, a_{n-1}, b_{n-1})$, we have eight unknowns $f_j; j = 0, \cdots, 7$. It can be verified that they must satisfy

$$\mathcal{K} = \begin{array}{l}(f'_1 + f'_4)(f'_1 + f'_5)(f'_1 + f'_6)(f'_1 + f'_7) \\ (f_3 \oplus f_4)(f_3 \oplus f_6)(f_7 \oplus f_4)(f_5 \oplus f_4) = 1\end{array}$$

□

The synchronous logic synthesis problem of Sect. (2) has thus been transformed into the following :

Combinational logic optimization problem.

given	an expression $\mathcal{K}(f_j)$ in terms of the $2^{n_i \times d} - 1$ entries f_j of f;
determine	the minimum cost function f such that $\mathcal{K} = 1$.

It is worth remarking on the differences between this problem and others previously considered in the literature. The classic theory of incompletely specified functions [14, 15] considers incomplete specifications in which each entry is either assigned a value, or is a *don't care*. In the present case, instead, assignments to different entries are "correlated" : for example, by looking at the expression $\mathcal{K}$ in Example (8), it is easy to see that f_4 and f_5 must always be assigned opposite values.

A first generalization to the classic theory (the minimization of Boolean Relations [22]) has been considered recently in the context of optimization of multiple-output combinational circuits [20]. It was shown in particular that for some multiple-output logic optimization problems correlations exist between assignments to the same entries of different incompletely specified functions. Note, however, that different entries of a single function (in our case, f_4 and f_5), could still, in that case, be chosen independently.

4 Finding minimum cost solutions.

In this section we consider an algorithm for solving the above combinational optimization problem. As mentioned, we consider two-level, sum-of-products realizations, and we build in particular upon the Quine-McCluskey algorithm for Boolean functions [14, 15, 16, 22]. To this purpose, we first extend the definitions of cube, implicant and prime of a Boolean function to the present combinational optimization problem.

Definition 2. *A* **cube** $c(\mathbf{x}_n, \cdots, \mathbf{x}_{n-d})$ *on the variables of* $\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}$ *is the product of some of such variables, in either true or complemented form. The variables appearing in c are termed the* **support** *of c. 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 there is no implicant c' of f that strictly contains c.* □

We represent a feasible solution as sum of implicants. If a cube c is part of a feasible solution f, then for each assignment of $\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}$ such that $c = 1$ it must also be $f = 1$. We call the set of entry variables f_j for which $c = 1$ the **span** SP_c of c. We denote by $\mathcal{K}_c$ the function obtained from $\mathcal{K}$ by assigning the value 1 to all the f_j's in SP_c. Checking whether a cube c is an implicant thus reduces to checking whether there exists a solution to $\mathcal{K}_c = 1$.

The support $SUPP_{\mathcal{K}}$ is the set of entries f_j appearing in $\mathcal{K}$. $SUPP_{\mathcal{K}}$ provides a first information on *don't care* conditions in the feasible solutions: since every solution to $\mathcal{K} = 1$ represents an assignment only to the entry variables f_j appearing in $SUPP_{\mathcal{K}}$, all the other entries of f will always be left unspecified. They consequently represent *don't care* conditions **common to all feasible solutions.**

Example 9. For the function $\mathcal{K}$ derived in Example (8), $SUPP_{\mathcal{K}} = \{f_1, f_3, f_4, f_5, f_6, f_7\}$. The entries f_0 and f_2 will be left unspecified in every solution of $\mathcal{K} = 1$, and therefore are a *don't care* common to all feasible solutions. Consider the cube $c = b'_n a_{n-1}$. We have $SP_c = \{f_2, f_3\}$ and $\mathcal{K}_c = f'_1 f'_4 f'_6 f_5 f_7$. The assignment $f_1 = 0, f_4 = 0, f_6 = 0, f_5 = 1, f_7 = 1$ clearly solves $\mathcal{K}_c = 1$; therefore c is an implicant. It can also be verified that, corresponding to that assignment of the f_j's, c cannot be expanded, and it is consequently a prime. □

4.1 Extraction of primes

A **minterm** is defined as a cube of minimum size, *i.e.* whose span contains a single entry $f_j \in SUPP_{\mathcal{K}}$.

Similarly to the case of ordinary two-level logic minimization, all primes can be extracted simultaneously by starting from the set of minterms and iteratively expanding them [15, 14, 16]. The loop for finding all primes is illustrated by the procedure $AllPrimes$. At the k^{th} step, all the implicants in the set I and produced at the previous iteration are first checked for primeness. Larger implicants are then produced by removing a literal from implicants from the previous iteration [15, 14, 22].

The routine $ExpandImplicants$ checks the satisfiability of $\mathcal{K}_c = 1$.

To verify the primeness of an implicant c, it is similarly necessary to check the existence of a feasible solution f for which c is prime. By definition, c is prime if there exists an assignment of the f_j's such that $\mathcal{K}_c = 1$ and $\mathcal{K}_{c_x} = 0$ for all cubes c_x obtained by **expanding** c, *i.e.* by removing a single literal from it. The check for primeness on c thus consists in determining a solution to

$$\mathcal{K}_c \prod_{c_x \in E_c} \mathcal{K}'_{c_x} = 1$$

where E_c denotes the set of cubes obtained by removing a single literal from c. The selection of primes is performed by the routine $PrimesOf$.

$AllPrimes(\mathcal{K}, S)$;
input: expression $\mathcal{K}$, set S of minterms;
output: set of primes;

$P = \phi$;
$I = S$;
for $(k = 1, k \leq n, \times d, k++)\{$
  $P = P \bigcup PrimesOf(I, \mathcal{K})$;
  $I = ExpandImplicants(I, \mathcal{K})$;
  $\}$;
return (P);

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

4.2 Covering Step.

Once the list of primes has been built, Petrick's method can be used to construct the subsequent covering problem [23, 20, 22]. Let N denote the total number of primes $c_1, \cdots, c_N$. The general solution is written as

$$f = \sum_{r=1}^{N} \alpha_r c_r,$$

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

The cost W of the solution is expressed by

$$W = \sum_{r=1}^{N} w_r \alpha_r$$

where w_r is the cost associated with each prime c_r, for example its number of literals [22].

Let $\mathbf{x}^j$ denote the j^{th} assignment (of dimension $n_i \times d$) to the variables $\mathbf{x}_n, \cdots, \mathbf{x}_{n-d}$ (*i.e.* $\mathbf{x}^0$ = 00...0, $\mathbf{x}^1$ = 00...1, ...). For each f_j in $SUPP_{\mathcal{K}}$, it must be

$f_j = \sum_{r=1}^{N} \alpha_r c_r(\mathbf{x}^j)$.

This equation expresses the entries f_j in terms of the parameters α_r. By substituting these expressions in $\mathcal{K}$, we obtain a new expression $\mathcal{K}_\alpha(\alpha_r; r = 1, \cdots, N)$ of the feasible solutions in terms of the variables α_r.

The synthesis problem is thus eventually transformed into that of finding the minimum cost assignment to the variables α_r such that $\mathcal{K}_\alpha = 1$, and it 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 true and complemented form in the conjunctive form of $\mathcal{K}_\alpha$, as shown by the following example.

Example 10. For the optimization problem of Example (8),

$$\begin{array}{llll} f_0 = & 0; & f_5 = & \alpha_4; \\ f_3 = & \alpha_2 + \alpha_3; & f_6 = & \alpha_1; \\ f_4 = & \alpha_1; & f_7 = & \alpha_2 + \alpha_4. \end{array}$$

□

Note in particular that the entry f_1 is not covered by any cube. The equation $\mathcal{K} = 1$ can now be rewritten as

$$\begin{aligned} \mathcal{K}_\alpha = \; & (1+\alpha'_1)(1+\alpha'_5)(1+\alpha'_6)(1+\alpha'_7) \\ & [\alpha_1 \oplus (\alpha_2+\alpha_3)][\alpha_1 \oplus (\alpha_2+\alpha_4)][\alpha_1 \oplus \alpha_4] = 1. \end{aligned}$$

This constraint can, for example, be reduced to a conjunctive normal form:

$$\begin{aligned} \mathcal{K}_\alpha = \; & (\alpha_1+\alpha_3+\alpha_5)(\alpha'_1+\alpha'_3)(\alpha'_1+\alpha'_5) \\ & (\alpha_1+\alpha_2+\alpha_4)(\alpha'_1+\alpha'_2)(\alpha'_1+\alpha'_4)(\alpha_1+\alpha_4) = 1. \end{aligned}$$

The minimum-cost solution to $\mathcal{K}_\alpha = 1$ is represented by $\alpha_1 = 1, \alpha_2 = \alpha_3 = \alpha_4 = \alpha_5 = 0$, corresponding to $f = b'_{n-1}$.

It is perhaps worth noting that it may be the case that a given recurrence equation does not have definite solutions. In this case no assignment of the entries f_j can satisfy the constraints imposed by the relation table, and the search of primes necessarily aborts. In the case of logic optimization, however, we already know the existence of at least one definite solution (corresponding to the original subnetwork), and therefore, given enough time, the synthesis procedure will always complete.

4.3 Experimental Results

We implemented in C the algorithms described in this paper, and tested them on standard synchronous logic benchmarks. Following the framework of combinational logic optimization [1], the circuit is partitioned into single-output synchronous subnetworks, each of which is then optimized using the algorithms outlined in this paper. The global feedback paths of the circuit were cut and treated as primary outputs. Optimization is then repeated until no improvement occurs.

The results obtained on a DEC 5000 workstation are shown in Table 4.3. In particular, the first four columns refer to the initial benchmark statistics, in terms of inputs, outputs, literal, and register counts, respectively. Columns *optl* and *optr* report the final number of literals and registers obtained, while the column labeled *cpu* shows the CPU time in seconds.

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 5: Experimental results for some logic optimization benchmarks.

5 Summary and Future Work.

In this paper, we have considered a novel formulation to the synthesis and optimization problems for synchronous networks. We have shown that, in order to fully capture the degrees of freedom for optimization, the terminal specifications of synchronous circuits are best provided in terms of **sets of execution traces**, rather than state diagrams.

In this paper, we used **Synchronous Recurrence Equations** to describe such trace sets. This description arises from the nature of the original specifications provided in this paper, in terms of netlists; the analysis of trace specifications on state-diagram level descriptions is the object of ongoing research.

The synthesis problem for a synchronous circuit was cast as that of finding the minimum-cost solution to such equations. We presented a two-step exact solution algorithm for such equations. 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 showed that the type of solutions achievable in this way are in general not reachable by existing approaches to synchronous logic optimization.

Preliminary experimental results indicate the feasibility of the approach for the hierarchical optimization of large synchronous networks; the development of heuristics for exact or approximate solutions to the binate covering problem represent, however, future progress necessary to improve its efficiency.

Currently, the global feedback function of the optimized network is not changed. This actually represents an unnecessary restriction to optimization: the feedback function can, in principle, be altered, as long as the observable terminal behavior of the entire network is not affected by this change. Further investigation on this aspect could result in algorithms leading to better quality optimization results.

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] 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.

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

[6] 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.

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

[8] 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.

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

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

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

[12] M. Damiani and G. De Micheli, "Synchronous Logic Synthesis: Circuit Specifications and Optimization Algorithms", *Proc. ISCAS 1990*, pp. 1566-1570.

[13] 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.

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

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.