Metamorphosis: State Assignment by Retiming and Re-encoding

Category : 2 (Sequential Logic Synthesis and Verification)

Authors

Balakrishnan Iyer[1]

Tel: (413)-545-0715

Fax: (413)-545-1993

Email: biyer@batchr.ecs.umass.edu

and

Maciej Ciesielski

Tel: (413)-545-0401

Fax: (413)-545-1993

Email: ciesiel@illiac.ecs.umass.edu

Department of Electrical & Computer Engineering

University of Massachusetts at Amherst

Amherst, MA 01003.

[1]**Presenter & Corresponding Author**

Metamorphosis: State Assignment by Retiming and Re-encoding

Category: 2 (Sequential Logic Synthesis and Verification)

ABSTRACT

This paper presents **Metamorphosis**[1] *– a novel technique for optimal state assignment targeting multi-level logic implementations. We present a new formulation and synthesis techniques for the state assignment problem based on controlled retiming and re-encoding of a symbolically represented finite state machine (FSM) represented initially with a one-hot code.* **Metamorphosis** *is a new design paradigm that integrates the structural and behavioral design methodologies in a synergistic fashion and leverages the additional degrees of freedom to the synthesis of optimal FSM. The proposed technique differs drastically from previous approaches in that the encoding process is guided directly by the cost function (optimization criterion) rather than speculative estimates of the encoding heuristics on the final design cost. We present efficient encoding algorithms for both unconstrained and bit-constrained encoding problems. Another novel feature of* **Metamorphosis** *is that it permits the exploration of the encoding search space over a variety of user-defined cost functions within an integrated CAD framework. Furthermore, the technique can easily incorporate additional user defined constraints on the encoding. In this paper we illustrate the technique through performance driven synthesis of FSM. We also outline the extensions to handle other cost metrics like sequential testability, area and power dissipation.*

[1]Webster defines Metamorphosis as "change of physical structure, form or substance".

Metamorphosis: State Assignment by Retiming and Re-encoding

1 Introduction and Overview

Despite the numerous advances in the field of synthesis and optimization of digital circuits and the wealth of literature on the subject, synthesis of synchronous sequential circuits remains an open problem [1]. One of the reasons contributing to this problem is that, to date, sequential optimization methods use one of the two mutually orthogonal modeling paradigms: 1) *Structural* (or network-based) models that represent circuits as synchronous networks, the interconnection of logic functions (or gates) and explicitly defined synchronizing elements (registers). Typical synthesis techniques that employ this model involve multi-level logic optimization[2] and retiming[3]. 2) *Behavioral* (or state-based) models, in which there is an explicit notion of a state, but only implicit notion of logic and registers; with these models sequential systems are specified by means of state transition graphs, tables, or state charts. Typical optimization problems in this domain involve state minimization and state encoding [4, 5].

It has been shown that, in principle, sequential circuits can be synthesized by a series of reencoding or retiming transformations with resynthesis [6, 7]. In this paper we consider state-based designs and concentrate on state encoding (or assignment) of finite state machines (FSM). State assignment is at the heart of the sequential synthesis problems, and despite large effort devoted to this problem no satisfactory solutions have been provided. The difficulty can be summarized with the following quote from the recent book by DeMicheli [1]: *"When considering state assignment for multi-level logic implementations, the quality of the results is still often unpredictable, because of the inability of current algorithms to forecast precisely the effects of the choice of the codes on the area and performance"*. With the exception of two-level circuits implemented as PLAs[8, 9], the state assignment algorithms are based on a prediction of the heuristically selected encoding on the subsequent logic optimization.

Metamorphosis overcomes the above problem by utilizing the directly measured value of the optimization criterion to guide the encoding process. This is made feasible by an entirely new formulation of the state assignment problem using a convenient matrix notation and a representation that relates the encoding to a one-hot code. This matrix formulation facilitates an efficient and reliable estimation of the optimization cost as a direct function of the encoding. The state assignment is achieved by a controlled reencoding/retiming of a symbolic representation of finite state machine represented initially with one-hot code. Thus, **Metamorphosis** integrates the *structural* and *behavioral* design paradigms in a synergistic fashion and leverages the additional degrees of freedom to the synthesis of optimal finite state machines.

The formulation and techniques presented in this paper define a flexible framework that can support a wide range of user-defined cost functions. Furthermore, the new formulation allows a rapid and reliable estimation of these measures more easily than previously possible. In this paper, we apply the formulation and techniques to performance optimization and suggest techniques to solve it efficiently. The results of the proposed technique can be directly benchmarked against the one-hot circuit, which is believed to give minimum delay implementations. Any improvement with respect to this reference point will therefore justify the computed state code.

2 Metamorphosis: Research Contributions

Metamorphosis is a design methodology that combines the *structural* and *behavioral* modeling paradigms to transform a given symbolic finite state machine into an circuit optimized to some cost function. The salient features of this technique are

1. A constructive encoding scheme that is driven by a user-specified cost function and *not* on speculative heuristic techniques.

2. An flexible and integrated CAD framework that permits the exploration of the design search space over a variety of user defined cost metrics and constraints.

3. A new and elegant formulation which facilitates reliable and rapid estimation of a variety of circuit quality measures.

4. Efficient and effective synthesis techniques for state encoding based on the new formulation.

5. Application of the technique to performance optimal state assignment.

3 Retiming vs Reencoding

Retiming is a recognized optimization technique for cycle-time minimization of synchronous sequential circuits [10, 3]. It is based on modifying the sequential structure of a circuit by relocating the state registers across its logic gates. Since retiming affects the next state functionality of the circuit, in effect altering its binary state encoding, it can be viewed as *reencoding* of its states. It is thus interesting to examine if a circuit with a given state encoding can be transformed into an equivalent circuit with another encoding by means of retiming. This question has been already raised in the literature and the following theorem, due to Malik et. al. [7], provides a partial answer to this problem: *Given a machine implementation M_1 with a given STG, and a state assignment S_1, it is always possible to derive a machine M_2 with the same STG, and a state assignment S_2 by applying only a series of resynthesis and retiming operations on M_1.*

The proof of this theorem can be outlined as follows: The combinational component N_1 of machine M_1 is appended with an identity logic block $I = C \cdot C^{-1}$, where C is a mapping between the states of M_1 and M_2, and C^{-1} is an inverse mapping. The state registers R_1, representing encoding S_1, are then retimed backward across C^{-1}, leading to a circuit M_2 with encoding S_2 and register set R_2, as shown in Fig. 1.

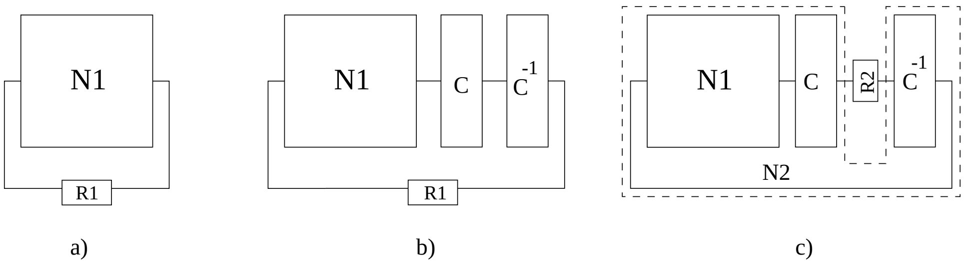

Figure 1: Reencoding by retiming and resynthesis: a) machine M_1, b) M_1 after resynthesis c) retiming leading to M_2

The resulting transformation of machine M_1 into M_2 can be represented symbolically as follows:

$$M_1 = \{N_1, R_1\} = \{[N_1|C|C^{-1}], R_1\} = \{[C^{-1}|N_1|C], R_2\} = \{N_2, R_2\} = M_2. \tag{1}$$

Due to one-to-one correspondence between the present-state and the next-state variables of the machine, this transformation can be viewed as a transformation of the combinational logic block N of the machine. We can then concentrate on restructuring the combinational logic only, without any concern for the registers:

$$N_1 = [N_1|C|C^{-1}] \rightarrow [C^{-1}|N_1|C] = N_2. \tag{2}$$

The cost of N can be easily related to that of M: for example, the delay of N determines the minimum clock-cycle of M, etc. While the two machines, M_1, M_2, are equivalent in terms of their synchronous behaviors, the corresponding combinational blocks, N_1, N_2, will, in general, have different functionality, design characteristics, and cost (area, delay, power, etc.). The goal is to find the *transformation logic* C that will make the reencoded circuit optimum according to some cost function; the above theorem seems to provide a means to do that. However, the retiming approach outlined above is limited to circuits with the *same state transition graph* and typically with the same encoding length. The reason is that both C and C^{-1} must exist, i.e., both must be Boolean functions. This condition is trivially satisfied when C is an injective (one-to-one) mapping, and in particular when the encoding length of the initial circuit is equal to the code length of the retimed circuit. Unfortunately, for arbitrary reencoding, function C (or C^{-1}) may not be injective, in which case C^{-1} (C) is not a function but a relation; as a result, the reencoding defined by this mapping cannot be implemented [2]. This limitation is a direct consequence of the requirement that the STG's of the original and the retimed circuit be identical.

The ultimate goal of sequential synthesis is to *find an encoding* (and hence a logic block C) that optimize the retimed/re-encoded circuit for a given cost function. The effective exploration of the encoding search space (to find the optimal FSM encoding) mandates more general reencoding techniques that can handle transformations involving circuits with different (but equivalent) state transition graphs (STG's), and different code lengths. To this end we propose a new retiming/re-encoding method that transforms a circuit with a given encoding into a circuit with an *arbitrary* encoding and code length, and an *equivalent*, but not necessarily identical, state transition graph. Eventually, this leads to a new encoding method described in Section 7.

4 Re-encoding by retiming of one-hot circuit

The proposed transformation of the given FSM into a re-encoded FSM will be achieved in two steps. For ease of exposition, we assume that we have been given the "desired" encoding matrix E. The methods for computing the "desired" encoding matrix which optimizes the design cost is discussed in Section 7. First the circuit is transformed into a network with one-hot code (where each state is assigned exactly one bit with value 1, other bits being 0). This is done by applying the identity transformation $I = H \cdot H^{-1}$, where H represents the mapping between the

[2]It has been shown in a recent work [6] that under certain conditions blocks C or C^{-1} need not be functions for the retiming to exist. These conditions involve cases of state minimization and state splitting, where logic blocks C or C^{-1} uniquely define internal logic functions, called *retimable functions*, across which the retiming of the network N can be performed.

initial state code and the one-hot code, and then by retiming the circuit across H. This can be represented as the following transformation of the original combinational network N into one-hot network N_H:

$$N = [N|H|H^{-1}] \rightarrow [H^{-1}|N|H] = N_H, \tag{3}$$

Then, the resulting circuit is reencoded to yield a circuit with the desired code E:

$$N_H = [N_H|C_H|C_H^{-1}] \rightarrow [C_H^{-1}|N_H|C_H] = N_E. \tag{4}$$

Here, C_H is the mapping between the one-hot encoding and the desired state code. It can be shown that both H and C_H are injective mappings, so that H^{-1} and C_H^{-1} are well defined for an arbitrary reencoding. Since the transformation into one-hot coded circuit is straightforward (function H is well defined), we concentrate here on the transformation of a one-hot circuit into a circuit with the desired encoding. The selection of the "desired" code which optimizes the design cost, which is our ultimate goal is discussed in Section 7.

5 Motivating Example

We will illustrate the re-encoding process by means of a simple example. Let the cost function to be minimized be circuit delay. For simplicity, we assume a unit delay model.

Consider the state machine (with outputs removed for simplicity) and its one-hot implementation, shown in Fig. 2. The circuit can be trivially obtained from the state transition table by replacing symbolic states S_i with one-hot code: $S_0 = 1000,\ S_1 = 0100,\ S_2 = 0010,\ S_3 = 0001$, (or by re-encoding with $I = H{\cdot}H^{-1}$) and deriving the next-state functions: $s_o' = (i_1{+}i_2){\cdot}S_0{+}S_3,\ s_1' = \bar{i_1} \cdot \bar{i_2} \cdot S_0$, etc. (See Fig. 2(b).)

Assume that we want to re-encode the states so that $S_0 = 00, S_1 = 10, S_2 = 01, S_3 = 11$. Such an encoding can be represented by a binary *encoding matrix* E, whose rows represent the state codes, and columns correspond to state bits. Matrix E uniquely defines logic C_H and its inverse:

$$E = \begin{bmatrix} 0\,0 \\ 1\,0 \\ 0\,1 \\ 1\,1 \end{bmatrix} \Rightarrow C_H = \begin{array}{|c|c|} \hline s_0 s_1 s_2 s_3 & r_1 r_2 \\ \hline 1\,0\,0\,0 & 0\,0 \\ 0\,1\,0\,0 & 1\,0 \\ 0\,0\,1\,0 & 0\,1 \\ 0\,0\,0\,1 & 1\,1 \\ \hline \end{array} \qquad C_H^{-1} = \begin{array}{|c|c|} \hline r_1 r_2 & s_0 s_1 s_2 s_3 \\ \hline 0\,0 & 1\,0\,0\,0 \\ 1\,0 & 0\,1\,0\,0 \\ 0\,1 & 0\,0\,1\,0 \\ 1\,1 & 0\,0\,0\,1 \\ \hline \end{array}$$

Fig. 3 shows a step-by-step reencoding transformation of the circuit. Fig. 3(a) shows the one-hot network N_H appended with an identity logic, $I = C_H C_H^{-1}$, and Fig. 3(b) the circuit after

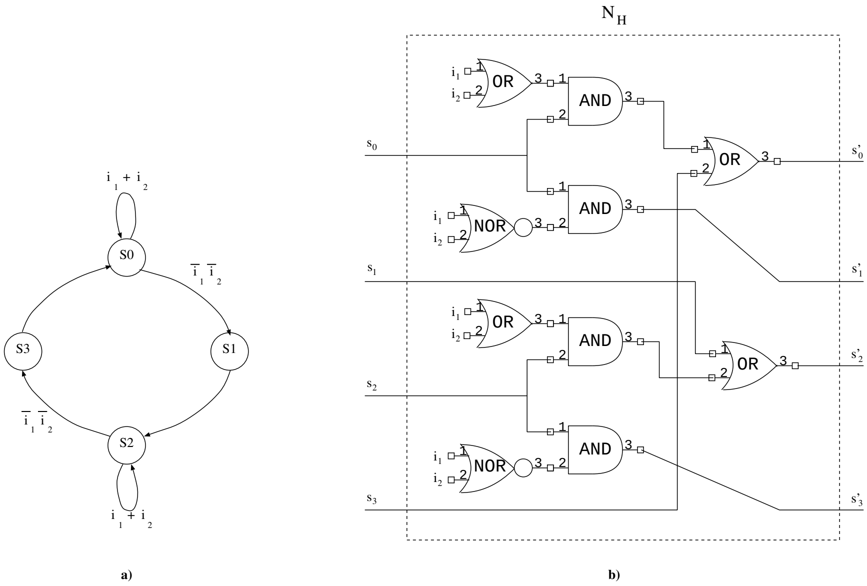

Figure 2: Machine with one-hot encoding: a) STG, b) one-hot network N_H

retiming across logic block C_H. Notice that this retiming transforms state space S, encoded with bits $\{s_0, s_1, s_2, s_3\}$, into a new space R, with bits $\{r_1, r_2\}$. Circuit in Fig. 3(c) is obtained after deleting the gates without any fanouts (e.g. s_0', s_1, etc.). Finally, logic simplification and remapping yields the circuit in Fig. 3(d). We can see that the final circuit has half the number of gates (5 gates) and smaller delay (2 gate delays) than the initial one-hot coded circuit (10 gates, 2 gate delays). This example illustrates that there exists a potential for improvement over the one-hot encoding both in terms of delay (performance) and area.

6 State Assignment: Optimal One-Hot Encoding

6.1 Matrix Formulation of the Re-encoding Process

In this section we assume that the encoding matrix E is given. The determination of the optimal encoding matrix E is the subject of Section 7.

Our formulation is based on the following matrix representation, similar to that used in classical circuit theory. The initial one-hot circuit N_H is represented by a square *transmission matrix* A_H, which describes its present-state to next-state behavior. The columns of the matrix are associated with symbolic next states and its rows with present states, S_i. An entry $A_H(i,j)$ is determined by an implicant of the next-state function associated with the state transition $S_i \to S_j$. Such an implicant has the form $\{I_k\ S_i \mid S_j\}$, where I_k is the primary input (predicate

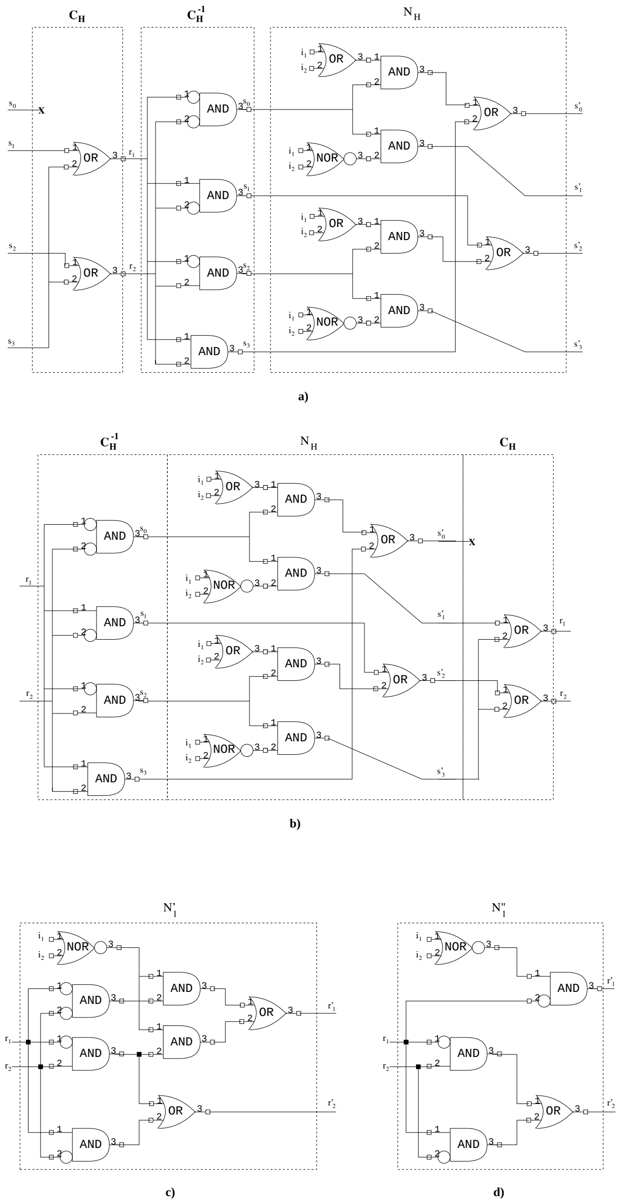

Figure 3: Reencoding of one-hot circuit: a) appending N_H with $I = C_H C_H^{-1}$; b) circuit after retiming across C_H; c) circuit after simplification; d) final optimized circuit

of the transition), represented as a Boolean expression in terms of the binary input variables; S_i and S_j are the present and the next state, respectively. An entry $A_H(i,j)$ is then simply defined as an input predicate I_k (in sum-of-products form) associated with the transition $S_i \rightarrow S_j$. Since in a deterministic machine there is only one such transition, $A_H(i,j)$ contains a single expression of the primary input variables. With this representation, an expression for next-state S_j can be computed as a logical sum (OR) of the entries in column j of A_H. For one-hot encoding this defines the next-state function for s'_j. In our example, $s'_0 = (i_1 + i_2) \cdot S_0 + S_3$, $s'_1 = \bar{i_1} \cdot \bar{i_2} \cdot S_0$, etc., and

$$A_H = \begin{bmatrix} (i_1 + i_2) & \bar{i_1} \cdot \bar{i_2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & (i_1 + i_2) & \bar{i_1} \cdot \bar{i_2} \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

Similarly, we can represent logic blocks C_H and C_H^{-1} in matrix notation. It turns out that they are both trivially associated with the encoding matrix E. To link logic C_H with matrix E, notice that an OR gate in C_H corresponds to a *column* of the encoding matrix; furthermore, the inputs to the OR gate are determined by the "1" entries in the corresponding column of E. In our example, the first column of E induces equation $r'_1 = s'_1 + s'_3$, and the second column $r'_2 = s'_2 + s'_3$ (compare it to the circuit in Fig. 3(c)).

By the same token, an AND gate in C_H^{-1} corresponds to a *row* of the encoding matrix E, which contains the binary encoding of the corresponding state. The 0 and 1 entries of a row determine the polarity of the inputs to the gate. The inputs to the AND gates can be viewed *minterms* of the present state vector r. C_H^{-1} can be expressed as a *row permutation matrix* E_x. It is a square binary matrix whose columns correspond to the states, and rows to the minterms of the state bits (in the case of incompletely specified encoding with don't cares, the minterms can be replaced by cubes so that some of the AND gates may have fewer inputs). $E_x(i,j) = 1$, if the binary code of S_j corresponds to minterm m_i, and it is 0 otherwise. For example, $S_0 = 00 = \bar{r_1}\bar{r_2}$ implies that the matrix has 1 in column 1 (for S_0) and row 1 (for $\bar{r_1}\bar{r_2}$), etc. The effect of the row permutation matrix E_x is to order the minterms by increasing order of their indices. In our example this leads to the following result:

$$E_x A_H E = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \end{bmatrix} \begin{bmatrix} (i_1+i_2) & \bar{i_1}\cdot\bar{i_2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & (i_1+i_2) & \bar{i_1}\cdot\bar{i_2} \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0\,0 \\ 1\,0 \\ 0\,1 \\ 1\,1 \end{bmatrix} = \tag{6}$$

$$\begin{bmatrix} \bar{i_1}\cdot\bar{i_2} & 0 \\ \bar{i_1}\cdot\bar{i_2} & \bar{i_1}\cdot\bar{i_2}+(i_1+i_2) \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = A_E$$

This representation is particularly valuable, since each entry $A_E(i,j)$ of the resulting matrix contains a logic expression (in SOP form) relating the next state variable r'_j to minterm $m_i(r)$ of the state vector r. From this, a logic path $r_i \rightarrow r'_j$ can be also constructed in order to compute/estimate the delay in the circuit. The next-state expression for r'_j can be obtained as a logical sum of the entries in column j of the matrix multiplied by the corresponding minterm. For example, $A_E(2,1) = \bar{i_1}\cdot\bar{i_2}$ means that $r'_1 = \bar{i_1}\cdot\bar{i_2}\cdot m_2(r) = \bar{i_1}\cdot\bar{i_2}\cdot\bar{r}_1\cdot\bar{r}_2$. Subsequently, the expression for r'_1 can be obtained as the sum $\bar{i_1}\cdot\bar{i_2}\cdot m_1(r)+\bar{i_1}\cdot\bar{i_2}\cdot m_2(r) = \bar{i_1}\cdot\bar{i_2}\bar{r}_1\bar{r}_2+\bar{i_1}\cdot\bar{i_2}\bar{r}_1\cdot r_2$; compare this equation to the circuit in Fig. 3(c). This expression can be further simplified to $r'_1 = \bar{i_1}\cdot\bar{i_2}\cdot\bar{r}_1 = (\overline{i_1+i_2})\cdot\bar{r}_1$, see Fig. 3(d). For the purpose of delay minimization each expression in the matrix can be independently simplified or used directly to estimate the cost (delay) of the encoding.

In summary, our approach to state assignment based on the reencoding of one-hot circuit can be cast as the following optimization problem:

Given a one-hot circuit represented by matrix A_H, find

$$min_E \; cost(E_x A_H E) \tag{7}$$

over all legal encodings E, possibly subject to some constraints on E.

In this formulation, $cost(A_E)$, depends on the metric chosen for the optimization. For delay optimization, $cost(A_E) = \max\,\{delay(A_E(i,j)\}$; it can be similarly defined for other cost functions. This formulation can be readily extended to handle primary outputs, which has not been shown here for the sake of simplicity.

6.2 Mathematical Formulation of Optimal Encoding

We recognize that the ultimate objective is to find a "desired" encoding (or, equivalently, a suitable logic C_H), which will minimize the design cost. This, in effect, is known as *minimum-cost state assignment* problem. We shall now formulate the state assignment as a reencoding of one-hot network, referred to as a *one-hot reencoding*, and examine methods to solve it.

In this section we present the algorithms for the performance driven synthesis of FSM. We will present the algorithms for the unconstrained and bit-constrained encoding problems. As discussed in Section 6.1, each of the entries a_{ij} in the matrix A_H is a Boolean expression in the primary inputs to the network and denotes the path from the present symbolic state s_i to the next symbolic state s'_j. The purpose of our optimization is to determine an encoding matrix E such that the maximum delay from the present reencoded states $\{r\}$ to the next reencoded states $\{r'\}$ is minimized. The optimization problem can be mathematically expressed as :

$$\min_{E} \max(\delta(E_x A_H E)) \tag{8}$$

where, δ is a cost operator that evaluates the delay of each Boolean expression in the matrix product.

6.2.1 Cost Function: Delay Estimation by OR Decomposition

Given an arbitrary Boolean expression in SOP form and the signal arrival times at the inputs, the delay will be computed by decomposing the expression into basic 2-input (N)AND and (N)OR gates such that the resulting expression tree has minimum depth. This is accomplished by: 1) generating an AND gate for each product term and the OR gate to sum the product terms; 2) inverting each AND gate into a NOR gate on inverted inputs; and 3) decomposing each OR (NOR) gate into a tree of 2-input OR gates; this step is done in a fashion similar to the construction of the Huffman coding tree, taking signal arrival times into account [11]. 4) finally, the 2-input NOR gates with inverted inputs are inverted to 2-input NAND's. During the construction of a tree with 2-input gates, the correct value of the longest path delay will be maintained by performing the decomposition in topological order starting from the input. Such a decomposition provides an upper bound on the mapped delay of the circuit; technology mapping onto a more realistic library of gates can only improve the delay due to the much larger variety of gates available in a typical library.

While exact value of the final delay may differ from our estimate, it is the relative delay and its monotonicity that is important at this stage of the technology independent synthesis. The **tradeoff** between accuracy of estimation and computation cost involved in delay estimation will also be investigated.

Other delay estimation methods similar to those used in the *speedup* algorithm of MIS[12], the quick factorization used in SIS[2] and those based on Lawler's clustering technique [13] can also be used. Our major goal is to have a reliable upper bound estimate by as fast a computation as possible, because it is a part of the extractable expression evaluation. A rapid and very approximate estimate of a sum-of-products (SOP) form is based on the following equation :

$$\delta = \Delta_G \lceil log_2\ N \rceil + \Delta_G \lceil log_2\ F_{max} \rceil + A_i \tag{9}$$

where, Δ_G is the delay of the two-input gate, N is the number of product terms, F_{max} is the fanin of the product term with the largest number of literals, and A_i is the arrival time of the latest arriving input.

The delay estimation algorithm based on *Huffman decomposition* is as outlined in Algorithm 6.1 :

Algorithm 6.1

```
Delay_Estimate(N, P, A)
{
   /* N = The given Boolean Network. */
   /* P = The set of inputs to the network. */
   /* A = The Arrival times of each of the Input. */
   N1 = collapse(N); /* Collapse the network N. */
   full_simplify(N1); /* Simplify the network. */
   delay = huffman_decomp(N1, 0, 2); /* Decompose the network into 2 input OR gates. */
   return(delay);
}/* end Delay_Estimate */
```

The above procedure is illustrated by means of a simple illustrative example in Fig. 4.

7 Constructive Methods to Encoding

The matrix representation described in this report facilitates the much needed evaluation of the encoding. For example, given an encoding matrix E, one can quickly compute the resulting

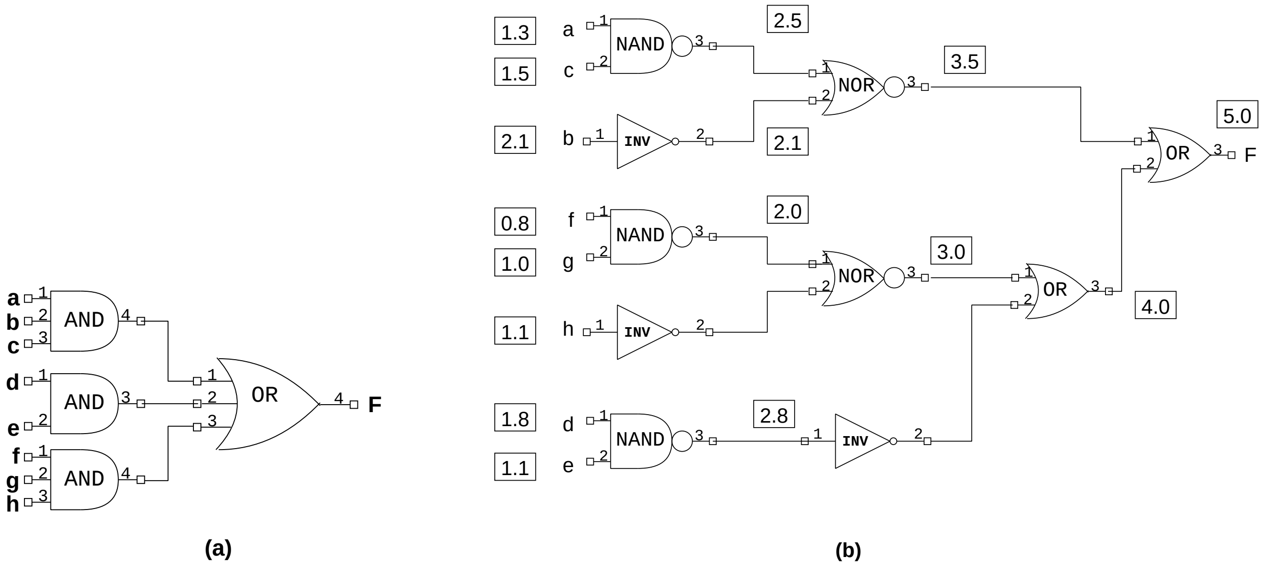

Figure 4: (a) A simple AND-OR circuit, (b) Delay computation based on Huffman OR-tree decomposition. The numbers in the boxes indicate the delay at the internal nodes of the decomposed circuit.

matrix A_E and evaluate the desired cost (area, delay, etc.). The following heuristics will be investigated to arrive at the optimal encoding matrix E: 1) *Column-based encoding*: column by column, i.e., bit by bit encoding. Generating column j of E corresponds to placing an OR gate at output r'_j. For a given column, the position of 1 entries determines to which outputs of N_H the OR gate is connected; the selection of these entries must lead to a maximum simplification of the resulting logic. 2) *Row-based encoding*: row by row, or state-by-state, encoding. Generating a row i of the encoding matrix corresponds to choosing the minterm of the encoding vector feeding the i^{th} AND gate at the input to N_H; or, equivalently, to assigning the polarity to the inputs of the gate. 3) *Mixed column/row encoding*, where the placement of the OR gates is interleaved with the polarity assignment of the AND gates. Determining the length of the state encoding and analyzing the tradeoff between the code length and the estimated delay will be an integral part of this encoding optimization process.

Our technique for minimizing the delay of a FSM is based on Boolean simplification. Boolean simplification is beneficial both in terms of reducing the area of implementation and enhancing performance. Boolean simplification in the context of re-encoding results from one of the following two ways:

1. The placement of an OR gate at an output r'_j may result in the simplification of the boolean expressions in the fanin of the OR gate. For example, in Fig. 3(b), the OR gate feeding r'_2 results in the following simplification $s_2 \cdot ((i_1 + i_2) + \bar{i_1} \cdot \bar{i_2}) = s_2$.

2. The assignment of the minterms feeding the AND gates at the input to N_H may result in the simplification of the nodes at the fanout of the AND gate. For example, in Fig. 3(c), simplification of the AND gates feeding the r'_1 occurs as follows $\bar{i_1} \cdot \bar{i_2} \cdot (\bar{r_1} \cdot \bar{r_2} + \bar{r_1} \cdot r_2) = \bar{i_1} \cdot \bar{i_2} \cdot \bar{r_1}$.

It has been our observation that, in general, the reduction in delay resulting from the placement of the OR gates is usually much larger than that obtained with the polarity assignment of inputs to the AND gates at the input to N_H. Based on this observation, we adopt a column based encoding scheme. However, if at any step there are two or more column encodings with the same "cost", we break the tie by choosing the assignment that will result in maximal simplification due to the polarity of the literals at the AND gate input.

Observe that the column based encoding involves the *bipartitioning* of the column into two groups – those with an entry of "1" and the others with an entry of "0". The outputs of N_H corresponding to a "1" entry form the fanin of the OR gate corresponding to the column. Encoding of a column also fixes the polarity of the corresponding re-encoded variable that drives the AND gates at the input of N_H. The observation that column based encoding involves bipartitioning leads to an elegant network flow based formulation of the encoding problem.

7.1 Flow Network Formulation

The performance driven encoding problem in Eq. 8 can be formulated as a flow network problem. The flow network is constructed as follows: each node in the flow network corresponds to a symbolic state in the one-hot FSM. The nodes in the flow network are connected by edges whose capacities correspond to reduction in delay by having the nodes in the same partition. Thus, the weight of edge between nodes u and v is given by :

$$w_{u,v} = \max(\delta(u), \delta(v)) - \delta(u + v) \tag{10}$$

Construction of a column of the encoding matrix is equivalent to bipartitioning its elements into 0's and 1's so as to minimize the delay of the longest path of the corresponding expression. While constructing the subsequent columns, the partitions induced by the construction of the previous columns are further bipartitioned. This process can be continued till all the columns of the encoding matrix have been constructed. The encoding of a column progressively constrains the encoding of the subsequent columns to ensure that the final codes are distinct. At the end of the encoding process, we should have distinct codes for each of the N states in the given symbolic one-hot FSM.

We use two different techniques for the solution of the bit-constrained (user specifies the maximum number of bits that can be used for encoding) and the unconstrained versions of the problem.

Unconstrained Encoding Problem **Ford Fulkerson** and **max-flow/min-cut** based algorithms [14, 15] – will be used for solutions with no constraints on the number of state bits. (hence no bound on the number of 0's and 1's in a column of the encoding matrix). This approach is also a good testbed for comparison with the one-hot encoding (which practically has no limit on the number of bits used for encoding). The unconstrained encoding problem gives a lower bound on the delay achievable for a given FSM and thus provides a bound for the bit constrained encoding problem.

Bit Constrained Encoding A user defined constraint on the number of encoding bits translates into a constraint on the sizes of the partitions of the flow network. This problem can be solved efficiently by using the **Kernighan & Lin** [16] algorithm or its variants. The algorithm involves pairwise exchange of the 0's and the 1's in the column of the encoding matrix E and evaluation of the new cost. The algorithm makes a series of such swaps (bipartitions) till the "optimum" encoding for the column has been computed. The KL algorithm has a hill-climbing capability which enables it to avoid being trapped in local minima. Once again, the construction of a column of the encoding matrix is equivalent to bipartitioning the elements of the column into 0s and 1s so as to minimize the delay of the longest path.

7.2 Other Metrics

In this section we outline extensions of the technique to handle other cost metrics like sequential testability, area of the implementation and power dissipation.

7.2.1 Sequential Testability

Notice that our formulation can readily accommodate additional *user-defined constraints*, such as input encoding constraints [17], or constraints leading to removal/avoidance of sequential false paths; the latter one can be used for both performance and testability improvement as demonstrated in our recent work [18] [19]. These constraints are specified as *dichotomies* and taken into account in the form of partial column encoding of E (placement of the OR gates).

7.2.2 Area

Area can be measured by the number of literals required in the implementation. For this case, the edge capacity in the flow network formulation is given by

$$w_{u,v} = n_{lit}(u) + n_{lit}(v) - n_{lit}(u+v) \tag{11}$$

where, $n_{lit}()$ is a cost function indicating the number of literals required to implement the given boolean expression. The savings in the number of literals may accrue from one of the following three ways (a) Simplification of the expressions due to placement of OR gates, (b) Simplification of the expressions due to the polarity assignment of the inputs of the AND gates, (c) Extraction of common sub-expressions (cubes and/or kernels) from the nodes in the network.

7.2.3 Power Dissipation

The minimization of power dissipated in sequential circuits is particularly hard for two reasons; power depends on the area of implementation and switching activity in the circuit, and both of these depend on the encoding. Tsui *et. al*[20] presented a power consumption model that takes into account the capacitive loading and the switching activity. In their paper the savings in power by extracting a common cube is measured by the number of literals saved weighted by the switching activity of the state bits. The flow network formulation proposed in this paper is easily amenable to such measurement.

8 Conclusions

In this paper, we have presented **Metamorphosis** – a new design methodology that integrates the structural and behavioral design paradigms in a synergistic fashion and leverages the additional degrees of freedom to the synthesis of optimal FSM. The proposed technique differs drastically from previously reported approaches in that it the encoding process is guided directly by the optimization criterion rather than speculative estimates of the encoding heuristics on the final implementation. This is made feasible by the elegant matrix notation that relates the encoding to the one-hot code. This formulation facilitates the efficient and reliable estimation of the optimization cost as a direct function of the encoding. We have presented efficient synthesis algorithms to construct the encoding matrix. The formulation and theory presented in this paper are quite general and can be applied to a variety of cost metrics like performance, sequential testability, area, power, etc.

We are presently in the process of implementing the above technique. The software will be interfaced to SIS[2]. We will have the results in time for the final version of the paper.

References

[1] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[2] E. Sentovich *et al.*, "Sis: A system for sequential circuit synthesis", Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Dept. of EECS, University of California, Berkeley., 1992.

[3] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle-time optimization", *IEEE Trans. on CAD*, vol. 10, pp. 63–73, Jan. 1991.

[4] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of finite state machines for optimal two-level logic implementations", *in IEEE Trans. on CAD*, vol. 9, pp. 905–924, Sep. 1990.

[5] S. Devadas, H-K.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of finite state machines targeting multi-level logic implementations", *in IEEE Trans. on CAD*, vol. 7, pp. 1290–1300, Dec. 1988.

[6] M. Ciesielski, "Functional retiming: A new approach to sequential synthesis and optimization", Technical Report TR-CSE-96-02, Department of Electrical & Computer Engineering, University of Massachusetts, Amherst., 1996.

[7] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques", *IEEE Trans. on CAD*, vol. 10, pp. 74–84, Jan. 1991.

[8] S. Yang and M.J. Ciesielski, "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization", *IEEE Trans. on Computer-Aided Design*, vol. 1, pp. 4–12, Jan. 1991.

[9] G. De Micheli, "Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-Level Logic Macros", *IEEE Trans. on CAD*, vol. CAD-5, pp. 597–616, Oct. 1986.

[10] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming", *in Third Caltech Conference on VLSI*, pp. 87–116, 1983.

[11] T-S Kim, "CAD Tools for Wave-Pipelined Circuit Design", *in Ph.D. Thesis, Dept. of ECE, University of Massachusetts at Amherst, MA 01003.*, Sep. 1995.

[12] K.J. Singh, A. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic", *in Intl. Conf. on CAD*, pp. 282–285, 1988.

[13] H.J. Touati, H. Savoj, and R.K. Brayton, "Delay optimization of combinational logic circuits through clustering and partial collapsing", *in Intl. Workshop on Logic Synthesis*, 1991.

[14] J.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton Uviversity Press, 1962.

[15] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[16] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *in The Bell System Technical Journal*, pp. 291–307, Feb. 1970.

[17] S. Yang and M.J. Ciesielski, "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization", *IEEE Trans. on Computer-Aided Design*, vol. 1, pp. 4–12, Jan. 1991.

[18] Z. Hasan and M. Ciesielski, "Elimination of multi-cycle false paths by state encoding", *in Proc. of European Design and Test Conference*, pp. 155–159, 1995.

[19] M. Ciesielski, "Sequential synthesis for testability: False-path-free encoding for improved atpg and partial scan test", Technical Report TR-CSE-96-01, Department of Electrical & Computer Engineering, University of Massachusetts, Amherst., 1996.

[20] C-Y. Tsui, M. Pedram, C-A. Chen, and A.M. Despain, "Low Power State Assignment Targeting Two- and Multi-level Logic Implementations", *in ICCAD*, pp. 82–87, Nov. 1994.