Metamorphosis: State Assignment by Retiming and Re-encoding

Balakrishnan Iyer Maciej Ciesielski
Department of Electrical & Computer Engineering
University of Massachusetts at Amherst
Amherst, MA 01003.

ABSTRACT

*This paper presents **Metamorphosis**[1] – a novel technique for optimal state assignment targeting multi-level logic implementations. We present an elegant matrix formulation and a graph partitioning based synthesis technique which permits both bit-constrained and unconstrained encoding of a symbolic finite state machine (FSM) represented initially with a one-hot code. Optimal state encoding is achieved by controlled retiming/re-encoding and resynthesis of the symbolic FSM. The synthesis is guided directly by the cost function (optimization criterion) rather than speculative estimates of the encoding heuristics on the final design cost. The technique is illustrated through performance driven synthesis of FSM and extensions to handle other cost metrics is outlined.*

1 Introduction and Overview

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[2, 3], the state assignment algorithms are based on a prediction of the heuristically selected encoding on the subsequent logic optimization.

Metamorphosis, on the other hand, utilizes a convenient matrix formulation which facilitates an efficient measurement of the optimization criterion for a given encoding. The symbolic FSM represented initially by an one-hot code is transformed into an optimally encoded FSM by controlled reencoding/retiming followed by resynthesis[4, 5]. Bit-constrained and unconstrained encoding problems are formulated as graph partitioning problems for which efficient algorithms exist[6, 7].

In this paper, we apply the formulation and techniques to performance driven synthesis of FSM 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 Retiming vs Reencoding

Retiming is a recognized optimization technique for cycle-time minimization of synchronous sequential circuits [8, 9]. 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. [5], 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.

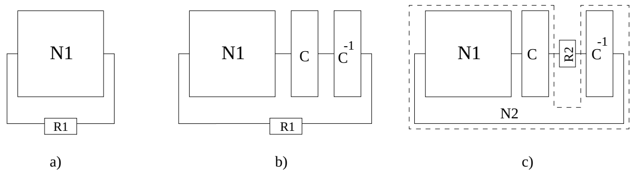

Figure 1: Reencoding by retiming and resynthesis: a) machine M_1, b) M_1 after resynthesis c) retiming leading to M_2

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

$$\begin{aligned} qM_1 = \{N_1, R_1\} = \{[N_1|C|C^{-1}], R_1\} = \\ \{[C^{-1}|N_1|C], R_2\} = \{N_2, R_2\} = M_2. \end{aligned} \quad (1)$$

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:

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

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 optimizes the retimed/re-encoded

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

[2] It has been shown in a recent work [4] 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.

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

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. This leads to the new encoding scheme described in Section 6.

3 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 6. 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 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:

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

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

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

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.

4 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

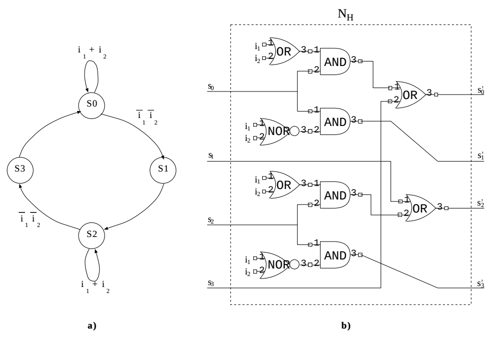

Figure 2: Machine with one-hot encoding: a) STG, b) one-hot network N_H

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).)

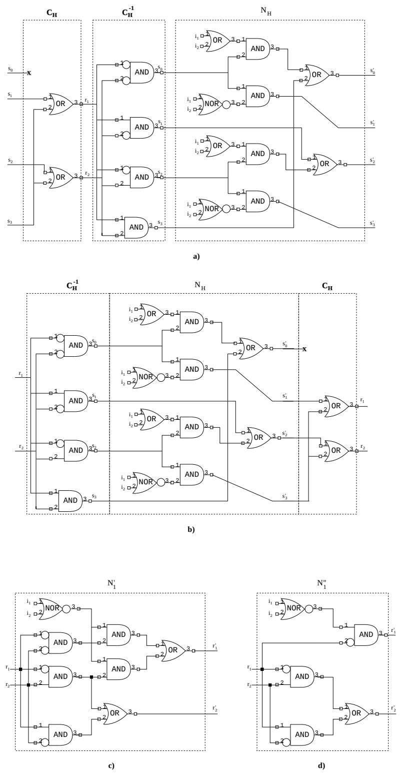

Figure 3: Reencoding of one-hot circuit: a) appending N_H with $I = C_H C_H^{-1}$; b) circuit after retiming across C_H; c) circuit after simplification; d) final optimized circuit

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:

$$E = \begin{bmatrix} 0\,0 \\ 1\,0 \\ 0\,1 \\ 1\,1 \end{bmatrix} \Rightarrow C_H =$$

$s_0 s_1 s_2 s_3$	$r_1 r_2$
1 0 0 0	0 0
0 1 0 0	1 0
0 0 1 0	0 1
0 0 0 1	1 1

$$C_H^{-1} =$$

$r_1 r_2$	$s_0 s_1 s_2 s_3$
0 0	1 0 0 0
1 0	0 1 0 0
0 1	0 0 1 0
1 1	0 0 0 1

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 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). 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, 3 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.

5 State Assignment by One-Hot Re-encoding

5.1 Matrix Formulation

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 $S_i \rightarrow S_j$. Such an implicant has the form $\{I_k\ S_i \mid S_j\}$, where I_k is the primary input (predicate 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. 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

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

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 as the *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:

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

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 $r'_{2,1} = \bar{i_1} \cdot \bar{i_2} \cdot m_2(r) = \bar{i_1} \cdot \bar{i_2} \cdot \bar{r_1} \cdot \bar{r_2}$. 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, 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

$$min_E\ cost(E_x A_H E) \quad (6)$$

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.

5.2 Cost Function: Delay Estimation

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 [10]. 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 technology independent synthesis. The **tradeoff** between accuracy of estimation and computation cost involved in delay estimation needs to be considered.

Other delay estimation methods similar to those used in the *speedup* algorithm of MIS[11], the quick factorization used in SIS[12] and those based on Lawler's clustering technique [13] can also be used. Delay estimates need to be computed as efficiently as possible, because it is a part of the extractable expression evaluation.

6 Constructive Methods to Encoding

The matrix representation facilitates the quick evaluation of the cost of a given encoding. The following heuristics can be employed 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.

Boolean simplification is beneficial both in terms of reducing the area of implementation and enhancing performance. Boolean simplification is exploited during re-encoding in 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.

6.1 Graph Partitioning Formulation

The performance driven encoding problem can be formulated as a graph partition problem. The graph is constructed as follows: each node in the graph corresponds to a symbolic state in the one-hot FSM. The nodes in the graph are connected by edges whose weights 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 :

$$w_{u,v} = \max(\delta(u), \delta(v)) - \delta(u + v) \qquad (7)$$

where, $\delta(u)$ and $\delta(v)$ denote the delay when u and v are in different partitions and $\delta(u + v)$ denotes the delay when u and v are in the same partition. 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 The absence of constraints on the number of encoding bits, translates to an unconstrained graph partitioning problem. This problem can be solved efficiently by $(n-1)$ applications of the max-flow/min-cut algorithm[6], where n is the number of symbolic states. 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).

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 graph. This problem can be solved efficiently by using the **Kernighan & Lin** [7] algorithm or its variants.

6.2 Other Metrics

In this section we outline extensions of the technique to handle other cost metrics.

Sequential Testability Notice that our formulation can readily accommodate additional *user-defined constraints*, such as input encoding constraints [2], 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 [14]. These constraints are specified as *dichotomies* and taken into account in the form of partial column encoding of E.

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

$$w_{u,v} = n_{lit}(u) + n_{lit}(v) - n_{lit}(u + v) \qquad (8)$$

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.

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*[15] 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 formulation proposed in this paper is easily amenable to such measurement.

7 Conclusions

In this paper, we have presented ***Metamorphosis*** – a new state encoding technique targeting multi-level logic implementations. The proposed encoding technique is guided directly by the optimization criterion rather than speculative estimates of the encoding heuristics on the final implementation. We have also presented an elegant matrix formulation and solution techniques based on graph partitioning heuristics.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] C.K. Cheng and T.C. Hu, "Maximum Concurrent Flows and Minimum Cuts", *Algorithmica*, vol. 8, pp. 233–249, 1992.

[7] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *in The Bell Sys. Tech. J.*, pp. 291–307, Feb. 1970.

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

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

[10] 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.

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

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

[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] 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.

[15] 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.