Reencoding for Cycle-Time Minimization under Fixed Encoding Length

Balakrishnan Iyer  
Department of Electrical & Computer Engineering, University of Massachusetts, Amherst, MA 01003.  
{biyer,ciesiel}@ecs.umass.edu

Abstract—This paper presents efficient reencoding and resynthesis algorithms for cycle-time minimization of multilevel implementations of synchronous finite state machines (FSMs) under a fixed encoding length. The proposed technique is applicable to both gate-level and technology independent network representations. We present two algorithms for identifying useful reencodings – one is based on Boolean cube representation applicable to technology independent synchronous networks and the other employs recursive learning techniques appropriate for gate netlists. We show that the proposed XOR/NOR based reencoding technique explores a sufficiently rich set of encodings to identify implementations with smaller cycle-times. The Boolean and structural interpretations of reencoding are explored and its relationship to isomorphic sequentially redundant faults is presented. We also show that the reencoded circuit always has a valid initial state and present a simple procedure to derive it. The effectiveness of the proposed technique is illustrated on a large set of benchmark circuits which indicates an average cycle-time improvement of 15.26% for a small area overhead of 3.56% over that of performance-driven combinational logic optimization.

I. INTRODUCTION

In this paper, we describe an iterative sequential reencoding and resynthesis technique for minimizing the cycle-time of multilevel circuit implementations of finite state machines (FSMs) while maintaining the same number of registers. It has been observed [1], [2] that the area increase in retiming and resynthesis is often due to the increase in the number of registers in the circuit. In our approach, this problem is avoided by restricting the reencoding to a fixed number of registers. While this limits the scope of area-performance tradeoffs, we show that a sufficiently rich choice of solutions is available for a large number of circuits.

Previous approaches to performance driven synthesis of FSMs can be classified into 2 categories: (i) Optimization and speed-up of the combinational logic component[3], [4] which ignores the interaction of gates across register boundaries. (ii) Gate-level retiming which involves moving registers across combinational logic blocks so as to minimize the cycle-time[5]. The drawback of gate-level retiming is that it is guided by the minimization of cycle-time based on a precomputed function of the location of the registers in the network and does not consider any prospective logic optimization. There have been several attempts to overcome the limitations of gate-level retiming[6], [1], [7] by combining retiming with combinational logic resynthesis. However, no satisfactory solutions have emerged – either the cycle-time improvement has been marginal or it comes at a prohibitive area overhead.

In an orthogonal direction multilevel and sequential logic optimization techniques that leverage the advances in automatic test pattern generation (ATPG) have been proposed[8], [2]. These approaches employ ATPG techniques as a Boolean reasoning engine. ATPG based redundancy elimination represents an efficient implicit technique to minimize a circuit with respect to a don't care set[9].

Reencoding transformations similar to those presented in this paper have been applied to reduce the size of the reduced ordered binary decision diagram (ROBDD)[10] representations of FSMs[11], [12]. They show that the XOR transformation is the most promising among the set of all reencoding transformations.

II. RETIMING AND RESYNTHESIS

Our approach is based on the following theorem due to Malik et. al.[6]:

**Theorem 1:** [6] Given a machine implementation \( M_1 \) with a STG \( G \), and a state assignment \( S_1 \), it is always possible to derive a machine \( M_2 \) with the same STG \( G \), and a state assignment \( S_2 \) by applying only a series of resynthesis and retiming operations on \( M_1 \).

The proof of the theorem is outlined in the transformations shown in Fig. 1. The combinational component of machine \( M_1 \) is augmented by an identity logic block \( I = C.C^{-1} \), where \( C \) is the mapping between the states of machine \( M_1 \) and \( M_2 \) and \( C^{-1} \) is the inverse mapping. The registers \( R_1 \) are then retimed backward across \( C^{-1} \), and the combinational logic resynthesized, leading to the machine \( M_2 \) with encoding \( S_2 \) and register set \( R_2 \). The block \( C \) is often referred to as the encoder block and the block \( C^{-1} \) is called the decoder block in the sequel. The retiming and resynthesis approach outlined above is limited to circuits with the same STG and with the same encoding length. The reason is that both \( C \) and \( C^{-1} \) must exist, i.e. they must be Boolean functions. This is trivially true when \( C \) is an injective (one-to-one) mapping, and in particular when the encoding lengths of machines \( M_1 \) and \( M_2 \) are identical. This is exactly the type of transformation considered in this paper. While the synchronous behavior of machines \( M_1 \) and \( M_2 \) are equivalent, their corresponding combinational blocks have different functionalities and cost (area, performance, power). In this paper, we propose techniques to find an encoder block \( C \) which on retiming and resynthesis minimizes the cycle-time of machine \( M_2 \).

The reader is referred to [13], [14] for the definitions and notations used in logic synthesis and testing. In the sequel, we assume simultaneous implementations using edge-triggered registers (D-flip-flops). In this paper we use redundancy removal techniques to simplify the circuit. It has been shown that in the presence of false paths, i.e. paths that cannot propagate signals under any circumstances, redundancy removal and Boolean simplification can actually increase the delay of the circuit by making the long false path active. Fortunately, redundancies can be removed from circuits with false paths with no adverse effect on delay but at the cost of area increase using the KMS algorithm[15]. Thus, without loss of generality, we assume that the longest topological path is true.

III. MOTIVATING EXAMPLE

In this section, we illustrate the problem by means of a simple motivating example. Consider the gate-level implementation of a FSM shown in Fig. 2(a).

![Figure 1](image1.png)

**Fig. 1.** Reencoding by Retiming and Resynthesis: (a) original machine \( M_1 \), (b) machine \( M_2 \) with an identity block \( C.C^{-1} \) appended, (c) Retiming across \( C^{-1} \) block, (d) Resynthesized machine \( M_2 \).

The underlying STG and the state encoding are also indicated. The bubbles at the inputs of the gates indicate inversion. Assume that the target library provides two and three input AND/OR gates with all possible input polarities and that each two input gate has one unit of delay and the three input gate has two units of delay. The two boxes in the gate-level representation are the registers, annotated with the next state lines \( Y_1 \) and \( Y_2 \). The present state lines are indicated by \( Y_1 \) and \( Y_2 \). The primary input (PI) to the circuit is \( x \) and the primary output (PO) of the circuit is \( z \). The circuit has a delay of 3 on the output node \( z \). Note that the cycle-time cannot be reduced below this value by conventional gate-level retiming since the critical-path is the input-output path. Moreover, the cycle-time cannot be improved by the "retiming bottleneck removal" method presented in [7].

Now, consider the same STG with a different encoding of the states: \( A \leftarrow 00, \ldots \)
\[ B \leftarrow 01, C \leftarrow 11 \text{ and } D \leftarrow 10. \] In the new encoding we have swapped the codes of states C and D. A resynthesized circuit under the new state assignment is shown in Fig. 3. The reencoded circuit has a delay of 2 units.

The above reencoding can be put in the retiming and resynthesis perspective of [6], by noting that the required swapping of encodings between states C and D can be achieved by appending two XOR gates on the next state lines of the original machine in Fig. 2 as shown in Fig. 4(a). The reader may verify that the first XOR transforms the code of state C from 10 to 11 while the reverse is true for state D. The second XOR exists to restore the state encodings of states C and D to their original encoding. Thus, the first XOR gate labeled C\(^{-1}\) corresponds to the encoder block shown in Fig. 1 and the second XOR gate labeled C\(^{+1}\) is the decoder block in Fig. 1. Next we retime backwards across the second XOR gate as shown in Fig. 4(b) and resynthesize the resulting combinational circuit to get the circuit in Fig. 3 with the same number of registers.

Clearly it is quite impossible even for reasonably small machines to try all possible reencodings. In the sequel, we will present efficient algorithms for discovering such encodings leading to smaller cycle-times. We show how we arrive at the above reencoding from two perspectives: One is based on Boolean techniques and the other is based on recursive learning based implications[16] on the gate-level netlist.

### A. Reencoding through Boolean Techniques

The Karnaugh maps (K-maps) for the next state functions \( x_1 \) and \( y_2 \) for the example in Fig. 2 is shown in the top row of Fig. 5. We use Karnaugh map for illustration purposes – our algorithm and implementation is not based on Karnaugh maps. As can be seen from the K-map for \( z \) in the top row of Fig. 5, we can drop the dependency of \( z \) on \( y_1 \) by swapping the columns \( y_{12} = 11 \) and \( y_{12} = 10 \). This corresponds to swapping codes 11 and 10 in the design. In other words, we present 11 in the new circuit whenever 10 was presented in the original circuit, and vice-versa. Obviously, this cannot be done in isolation – the corresponding columns have to be swapped in the K-maps corresponding to \( y_1 \) and \( y_2 \) as indicated by the bidirectional arrows in Fig. 5. After the swapping of the rows indicated by the arrows, we have the K-map of \( z \) in the new circuit shown as \( z_1 \) in Fig. 5. The K-map \( Y_{12} \) in the new circuit remains unchanged from that of \( y_1 \) after swapping the columns. However, computing the K-map corresponding to \( Y_2 \) in the new circuit is a little more involved. Swapping the columns in the K-map of \( Y_2 \) gives the K-map denoted \( Y_{22} \) in the figure. The new K-map \( Y_{22} \) is computed as

\[
Y_{22} = Y_{22} \oplus Y_{10}
\] (1)

To see why this is the case, note that the swapping of the columns in the original K-maps is synonymous to swapping the corresponding present state inputs in the original state transition table (STT). In other words, we have modified the circuit such that whenever the registers have a value of 11(10) a value of 10(11) is presented to the original combinational block. However, in order to leave the synchronous behavior of the circuit unaltered we have to ensure that whenever a value of 11 is produced at the next-state lines the value stored in the registers is modified to 10 and vice-versa. This reflects the change in the encoding of the states C and D in the STG. This is achieved by the XOR operation in Eq. 1. The K-maps for the reencoded machine are shown in the bottom row of Fig. 5.

### B. Reencoding through Recursive Learning

The latest arriving signals on the critical path to \( z \) in the circuit of Fig. 2(a) are \( y_1 \) and \( y_2 \). If we can make a stuck-at fault on any of these is undetectable and hence redundant then the circuit can be simplified by removing the redundancy. Since we are concentrating on stuck-at faults on the latest arriving signals, any simplification due to redundancy of these faults would lead to a new circuit with delay at least as small as the original circuit if not smaller. Consider a stuck-at-0 fault \( F \) on the line \( L_1 \) in the original circuit of Fig. 2(a). This fault is indeed testable if it cannot be propagated to the output \( z \) for any and all possible values of the primary input \( x \). Since an input of \( x = 0 \) would trivially block the propagation of the fault, we will only consider the interesting case when \( x = 1 \). Also for any possible value of the present-state line \( y_2 \), the value of \( D \) should not be propagated to the output \( z \), i.e. it must be a 0 or 1 for both faulty and fault-free circuits. To see why this is the case, note that the swapping of the columns in the circuit of Fig. 2(a) is computed as

\[
Y_{22} = Y_{22} \oplus Y_{10}
\] (1)

To see why this is the case, note that the swapping of the columns in the original K-maps is synonymous to swapping the corresponding present state inputs in the original state transition table (STT). In other words, we have modified the circuit such that whenever the registers have a value of 11(10) a value of 10(11) is presented to the original combinational block. However, in order to leave the synchronous behavior of the circuit unaltered we have to ensure that whenever a value of 11 is produced at the next-state lines the value stored in the registers is modified to 10 and vice-versa. This reflects the change in the encoding of the states C and D in the STG. This is achieved by the XOR operation in Eq. 1. The K-maps for the reencoded machine are shown in the bottom row of Fig. 5.

The STG of the reencoded FSM is identical to that of the original FSM except for a relabeling of the states with different encodings. In this light, reencoding redundancy removal and resynthesis leads us to the final reencoded circuit shown in Fig. 3.

### IV. REENCODING ALGORITHMS

For a design with \( n \) register variables, there are \( 2^n \) possible encodings and the problem of finding the globally optimal encoding is intractable. In this section we present iterative algorithms to discover encodings resulting in smaller cycle-times. Each step in the iteration involves reencoding one of the latch variables by using the previously explained XOR/XNOR transforms. Each of these steps can be seen as a two-bit reencoding[11]. Our algorithms for reencoding are based on the following observations:

1. The number of minterms in the on-set of the functions computing the next-state values and the primary outputs remains invariant under the reencoding.
2. Interchange of minterms and cubes corresponding to different primary input alphabets (transition predicates) is not permitted. It can be shown that such interchanges lead to a increase in the number of registers in the design[18]. Thus, the...
permissible reencoding transformations are restricted to swaps in the present-state bit locations in the K-maps. In the recursive learning paradigm of Section III-B, this translates to the observation that the redundancy of an injected fault does not require a D or a T on any of the primary inputs.

3. In the course of reencoding we may swap the code assigned to a state with an unused code, i.e., one of the unused codes is now used and one of the codes in the used set becomes an unused code. However, the unused state codes are typically used as don’t cares in the optimization of the combinational logic. After the code swap the original don’t care set used to optimize the circuit is partially invalidated and has to be replaced by a new don’t care set. To ensure design correctness we need to propagate these changes throughout the design. We assume that the reachable state set (used codes) are available as part of the design specification as in SIS[19] or have been computed using implicit state traversal techniques[13], [20].

It has been shown in [11] that the iterative application of the XOR/XNOR transforms provides a good basis for the design of effective reencodings. They use counting arguments to prove the following result:

**Theorem 2:** [11] The number of possible encoding transformations, t(n) achieved by the iterative application of the XOR/XNOR is given by:

\[
t(n) = \prod_{i=0}^{n-1} (2^n - 2^n)
\]

where, \( n \) is the number of bits in the encoding.

Although, this is much smaller than the number of all possible reencodings, the XOR/XNOR operators provide a rich set of reencoding transformations. In fact, all of the 4!(= 24) one-to-one (bijective) mappings over 2 Boolean variables can be expressed as a composition of 3 basic operators: (i) the XOR/XNOR, operation over the 2 variables (ii) the register-variable permutation (or renaming), and (iii) the identity (no reencoding) operator[11]. Of these, the register renaming and identity transforms have no effect on the cycle-time of the implementation. Thus, the XOR/XNOR transform is the only possible one-to-one mapping over two Boolean variables. This is expressed in the following theorem.

**Theorem 3:** An encoding/decoding function pair implementing a two-bit reencoding of a synchronous network is valid if both members of the pair take one of the following two forms:

\[
y_k = f(Y) \cdot (y_k \oplus y_l)
\]

OR

\[
y_k = f(Y) \cdot (y_k \lor y_l)
\]

where, \( y_k \) \( y_l \) are register variables and \( f(Y) \) is some Boolean function, with \( Y = \{y| l \neq [k, l]\}\).

**Proof:** The proof follows from the following observations: the function \( f(Y) \) in Eq. 3, 4 serves as a “restriction” which limits the reencoding to states whose encodings satisfy the function \( f(Y) \). Thus, without loss of generality we can concentrate on the XOR/XNOR forms involving \( y_k \) and \( y_l \) in the above equations. The rest of the proof follows from previous arguments.

Note that this theorem provides only the necessary condition – in fact, there are many reencodings not explored by the proposed transform. However, the above theorem constitutes a necessary and sufficient condition when the original STG is reduced, has 2^n states and is encoded in \( n \) (minimum number) bits.

A. Algorithm based on Boolean Techniques

The algorithm is based on “truth-table permutations” to cluster the cubes used to compute the next-state and primary outputs. This can be computed in a straightforward manner on a ROBDD representation of the functions[12]. Next we iterate over the cubes in the next-state/primary output function(s) with the largest delay. In each iteration we try to merge the cubes and evaluate the impact of the transform on the circuit delay using the user specified delay model. If the transformation reduces the delay then it is accepted. The process is iterated until no improvement in delay results. Convergence of the procedure is ensured since delay is strictly decreasing and the slack on non-critical paths are decreasing. Thus, the process tends to equalize the delay of all paths. If the don’t care conditions are changed then the don’t care set is updated and the network is recompiled using the new don’t care set. The algorithm is outlined in Algorithm 1.

B. Algorithm based on Recursive Learning

The algorithm based on the ATPG technique using recursive learning as the implication engine is shown in Algorithm 2. The algorithm proceeds by identifying the stems on the critical path in the network. Stick-at-0 and stuck-at-1 faults are induced on these stems and recursive learning is used to learn the conditions for the redundancy of these faults. Stick-at-valuing control faults on these stems are attempted first failing which non-controlling values are tried. If the fault is deemed redundant for all primary input values then the circuit is transformed and recompiled using redundancy removal. If this involves swapping codes with a previously unused state code then part of the old don’t care set is invalidated. Optimization under the new don’t care network is accomplished using ATPG techniques presented in [9].

**Algorithm 1:** Reencode machine M using Boolean Techniques.

**Algorithm 2:** Reencode machine M using Recursive Learning.

**V. INITIAL STATE COMPUTATION**

Since we perform backward retiming across the decoder block we have to ensure that the retimed circuit has a valid initial state. The following theorem proves that a valid equivalent initial state always exists for the circuits produced by the proposed technique.

**Theorem 4:** Given an initial implementation \( M \) with initial state \( S_0 \), the new implementation \( M' \) produced by the proposed technique always has a valid initial state \( S_0' \). If the initial implementation \( M \) has a synchronizing (initialization) sequence that brings the machine into a known starting state then the same initializing sequence would bring the machine into the same starting state.

**Proof:** Refer to [18] for the proof.

In the proposed technique, only backward retimings are performed across the gates and fanout stems thus satisfying the conditions for 0-cycle safe replaceability[21].

**VI. RESULTS**

We implemented a prototype version of the proposed algorithms within the SIS[19] framework. The results for the proposed approach on the LGSynth89 and LGSynth91 benchmark suites are presented in Table I. As explained earlier in the paper, we leverage the existing techniques for performance driven synthesis of combinational logic blocks[3] and available in SIS[19]. Thus, in the sequel, we present the area overheads and the speedup achieved by the proposed technique over and above that achieved by performance driven synthesis of the combinational portion[3]. The initial circuit netlist was generated by minimizing the state machine using stamina[22], state assignment using jedi[23], and logic optimization using SIS[19] (using the script delaying) and mapped onto the lib2 library. The critical path(s) of the mapped circuit was traced and reencoding and resynthesis was performed to reduce the delay of the critical path(s). The resulting circuit was remapped for performance onto the same target library. Gate-level retiming was done using the retime command in SIS.

The results for the proposed technique, in general, shows a super-linear improvement in performance with respect to the area increase. In summary, the proposed approach achieves an average speedup of 14.91% in the combinational delay and 15.26% in cycle-time with an area overhead of 3.56% over that of circuits synthesized with script delay. Thus, the proposed technique is effective in improving the performance of state machine implementations with a modest increase in the silicon area. Also, the technique compares very favorably, both in terms of the area overhead and cycle-time, with gate-level retiming techniques that have been traditionally employed to improve performance.
TABLE I

RESULTS FOR REENCODING

<table>
<thead>
<tr>
<th>FSM Name</th>
<th>$[s]$</th>
<th>$[t]$</th>
<th>$[r]$</th>
<th>Script delay</th>
<th>Reencoded delay</th>
<th>$[T]$</th>
<th>$[S]$</th>
<th>$[S]_d$</th>
<th>$[D]$</th>
<th>SD + Retime</th>
<th>Ret. + Retime</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibara</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>71776</td>
<td>7.13</td>
<td>9992</td>
<td>6.24</td>
<td>8.63</td>
<td>-3.77</td>
<td>14.38</td>
<td>14.83</td>
</tr>
<tr>
<td>ibibo</td>
<td>13</td>
<td>11</td>
<td>7</td>
<td>122444</td>
<td>9.66</td>
<td>22222</td>
<td>7.86</td>
<td>11.58</td>
<td>-2.07</td>
<td>24.16</td>
<td>22.60</td>
</tr>
<tr>
<td>ibdas</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>35728</td>
<td>4.34</td>
<td>28888</td>
<td>4.09</td>
<td>5.62</td>
<td>-1.28</td>
<td>6.91</td>
<td>6.61</td>
</tr>
<tr>
<td>becoun</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>4600</td>
<td>4.45</td>
<td>600</td>
<td>4.45</td>
<td>6.27</td>
<td>-0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>cne</td>
<td>16</td>
<td>11</td>
<td>7</td>
<td>275918</td>
<td>12.69</td>
<td>29972</td>
<td>8.37</td>
<td>11.80</td>
<td>-2.93</td>
<td>9.31</td>
<td>8.40</td>
</tr>
<tr>
<td>qik1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
<tr>
<td>qik2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
<tr>
<td>sclt</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
<tr>
<td>cne</td>
<td>16</td>
<td>11</td>
<td>7</td>
<td>275918</td>
<td>12.69</td>
<td>29972</td>
<td>8.37</td>
<td>11.80</td>
<td>-2.93</td>
<td>9.31</td>
<td>8.40</td>
</tr>
<tr>
<td>qik1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
<tr>
<td>qik2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
<tr>
<td>sclt</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>54912</td>
<td>11.19</td>
<td>5408</td>
<td>9.28</td>
<td>12.10</td>
<td>-2.07</td>
<td>20.97</td>
<td>21.97</td>
</tr>
</tbody>
</table>

Legend: $[s]$ = # of states, $[o]$ = # of outputs and $[r]$ = # of registers. Columns 5 through 7 indicate the area, delay and cycle-time achieved with script.delay. Columns 8 through 10 give the corresponding values for the reencoding scheme. Column 11 gives the # area increase over script.delay due to reencoding (negative values indicate decrease in area). Columns 12 and 13 give the % speedup in combinational delay and cycle-time, respectively. Columns 14 through 16 indicate the # of registers, area and cycle-time after retiming the circuits produced by script.delay. Columns 17 through 19 give the corresponding numbers for retiming the circuits produced by reencoding. A “<” in the last 6 columns indicates that retiming failed to improve the cycle-time.

VII. CONCLUSIONS

In this paper we have presented reencoding and resynthesis algorithms based on XOR/XNOR reencoding transformations for cycle-time minimization of FSM implementations. The proposed technique satisfies the constraint that the number of registers in the circuit remain constant. We also proved that a valid initial state always exists for the reencoded circuit. Techniques to identify potentially useful reencodings were presented for synchronous network representations as well as gate-level netlist representations. The Boolean and structural interpretations of reencoding were explored and its relationship to isomorphic sequentially redundant faults was presented. In summary, the reencoding and resynthesis for performance can be seen as a technique for “slack redistribution” – reducing the depth of long paths while increasing the depth of short paths. The effectiveness of the technique is attested by the results on the benchmark circuits.

REFERENCES