General Decomposition of Sequential Machines: Relationships to State Assignment

Srinivas Devadas

Abstract

In this paper, we present new techniques for state assignment of finite state machines based on state machine decomposition algorithms.

A finite state machine can be decomposed into smaller interacting machines so as to optimize area and performance of the eventual logic implementation. A recently proposed form of decomposition, which has been shown to be superior to previous decomposition methods, involves identifying subroutines or factors in the original machine and extracting these factors to produce factored and factoring machines.

Optimal state assignment corresponds to finding an optimal multiple general decomposition of a finite state machine. We present state assignment techniques targeting two-level and multi-level logic implementations based on factorization algorithms followed by state assignment algorithms. For the two-level case, we prove that one-hot encoding a non-trivially factored machine is guaranteed to produce a better result than one-hot encoding the original machine. Experimental results indicate that this technique of factorization followed by state assignment is superior to previous state assignment techniques for large sequential machines, when targeting either two-level or multi-level implementations.
Acknowledgements

To appear in Proceedings, 26th Design Automation Conference, Las Vegas, Nevada, June 26-29, 1989. This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825.

Author Information

Devadas: Electrical Engineering and Computer Science, Room 36-848, MIT, Cambridge, MA 02139. (617) 253-0454.

Copyright © 1989 MIT. Memos in this series are for use inside MIT and are not considered to be published merely by virtue of appearing in this series. This copy is for private circulation only and may not be further copied or distributed, except for government purposes, if the paper acknowledges U.S. Government sponsorship. References to this work should be either to the published version, if any, or in the form “private communication.” For information about the ideas expressed herein, contact the author directly. For information about this series, contact Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; (617) 253-8138.
General Decomposition of Sequential Machines: Relationships to State Assignment

Srinivas Devadas
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

Abstract

In this paper, we present new techniques for state assignment of finite state machines based on state machine decomposition algorithms.

A finite state machine can be decomposed into smaller interacting machines so as to optimize area and performance of the eventual logic implementation. A recently proposed form of decomposition, which has been shown to be superior to previous decomposition methods, involves identifying subroutines or factors in the original machine and extracting these factors to produce factored and factoring machines.

Optimal state assignment corresponds to finding an optimal multiple general decomposition of a finite state machine. We present state assignment techniques targeting two-level and multi-level logic implementations based on factorization algorithms followed by state assignment algorithms. For the two-level case, we prove that one-hot encoding a non-trivially factored machine is guaranteed to produce a better result than one-hot encoding the original machine. Experimental results indicate that this technique of factorization followed by state assignment is superior to previous state assignment techniques for large sequential machines, when targeting either two-level or multi-level implementations.

1 Introduction

The problem of optimal state assignment involves finding a binary encoding of internal states in a finite state machine so as to produce a minimum area logic implementation after encoding and logic optimization. Several techniques for state assignment have been proposed in the past; techniques targeting two-level or sum-of-products implementations (e.g., [4], [7], [6]) and more recently, techniques targeting multi-level combinational logic implementations [2] [9]. While the techniques targeting two-level implementations attempt to minimize the number of product terms in the eventual implementation, the techniques targeting multi-level logic attempt to minimize the number of literals in the final implementation.

In [4], the use of multiple-valued minimization to find an optimal input (present state) encoding and therefore good state assignment was introduced and results better than previously proposed techniques were presented. However, one failing of this strategy is that it ignores the next state space/field of the finite state machine completely, sometimes resulting in a sub-optimal state assignment. The problem was partially alleviated in [7], via the use of heuristic output (next state) encoding algorithms, but the problem of effective integration between the input and output encoding steps failed to be solved. An encoding that satisfies the constraints generated by both the input and output encoding steps is not guaranteed to be found. Similar problems afflict the early techniques of state assignment that target multi-level logic implementations [2].

It is often convenient to realize a sequential circuit as an interconnection of two or more subcircuits for area and performance reasons. Heuristics for state assignment and techniques for logic optimization in sequential circuit synthesis work better for small circuits as opposed to large ones. If a good decomposition can be found, generally speaking, smaller areas for decomposed circuits over a single lumped circuit will result. The decomposition may be attractive from a performance point of view as well. The decomposed circuits can be clocked faster than the original machine due to smaller critical path delays. Of course, a truly optimum state assignment algorithm followed by an optimum logic optimization step will not work better for a decomposed circuit as opposed to a lumped circuit.

Decomposition methods can be classified into three main categories — parallel, cascade and general decompositions, corresponding to no interaction, unidirectional interaction, and bi-directional interaction between the decomposed submachines.

The decomposition of sequential machines was first treated in a formal way by Hartmanis [5] in 1960. The work was expanded in [6] and by others. Unfortunately, cascade decomposition has limited use in the design of modern finite state machines. For example, specifications of centralized controllers in microprocessor chips do not usually have good cascade decompositions. In [3], factoring algorithms for finding general decompositions of machines were proposed. These algorithms identify sets of states in the original machine with similar internal relationships and represent these sets of states by a call to a factoring submachine to produce a factored/decomposed submachine. For large machines, the areas of the decomposed realizations were smaller than the lumped circuits.

If factorizing prior to encoding and optimizing a given machine produces a better result than encoding and optimization without decomposition, it means that the heuristic encoding step produced a suboptimal result in the latter case. In fact, given the decomposed machines, an encoding can be constructed for the original machine which produces a result at least of the same quality as the former case. The problem of optimal state assignment corresponds to finding an optimal multiple general decomposition of a machine (by multiple decomposition we mean decomposition in several components rather than just two).

In this paper, we present new techniques for state assignment of finite state machines, targeting both two-
level and multi-level logic implementations, based on state machine factorization followed by the state assignment techniques of [4] and [2]. If the machine can be non-trivially factored, we prove that a better state assignment than one-hot encoding is guaranteed using these techniques in the two-level case. This means that factorizing a machine prior to using a KISS-style [4] algorithm has a upper bound on the number of resulting product terms that is lower than the upper bound with the factors isolated on the original machine. Since this upper bound when using KISS-style algorithms is a tight one, lowering it almost always improves the results. While we can only prove a weaker result for the more complicated multi-level case, experimental results indicate that decomposition followed by state assignment produces superior results for large sequential machines, in this case as well. The initial factorization results in effective exploitation of the relationships between the input (present state) space and output (next state) space in the decomposed machines for both the two-level and multi-level cases.

2 Preliminaries

For an example factorization of a machine, the reader is referred to [3]. Any \( N_R \) disjoint sets of \( N_F \) states can be extracted as occurrences of a factor and the decomposition performed to obtain to submachines \( M_1 \) and \( M_2 \). The complexity of the decomposed submachines is profoundly affected by the choice of the factor. The transition edge relationships between the \( N_R \) sets of states play an important role in determining the quality of a factor. If exactly similar transition edge relationships exist between these sets of states, factorization will result in the smallest possible number of transition edges in the decomposed submachines. The flow of state information between the two machines, \( M_1 \) and \( M_2 \), will be minimal and an economical realization will result. However, if the sets of states corresponding to the occurrences of the factor have dissimilar transition edge relationships, the resulting submachines will be complex, i.e. with a large number of transition edges dependent on state information from the other machine.

We now present definitions which will be used to describe the notions of exact and ideal factorization (factors). The object being defined appears in bold type.

A factor is \( N_R \) \( \geq 1 \) sets of states and all fanout edges from these sets of states in the given machine. Each set of states is called an occurrence of the factor \( F \) and is denoted \( O_F \). The maximum number of states in any of the \( N_R \) occurrences of \( F \) is denoted \( N_F \) \( \geq 2 \).

A transition edge in the occurrence of a factor, \( O_F \), is an internal edge if it fans into and fans out of states within \( O_F \). An exit state in any \( O_F \) is one which has no internal fanout edges. A state in \( O_F \) is an internal state if all edges from the state fan into states within \( O_F \) alone, i.e. all fanout edges from the state are internal edges. An entry state in any \( O_F \) is one with no internal fanin edges. The fanin edges into a \( O_F \) are denoted \( \text{fin}(i) \). The fanout edges out of a \( O_F \) are denoted \( \text{fout}(i) \). The external edges outside of any factor occurrence are denoted \( \text{EXT} \).

Given two occurrences of a factor, \( O_F^1 \) and \( O_F^2 \), a state correspondence pair is \( (\xi_1, \xi_2) \), \( \xi_1 \in O_F^1 \), \( \xi_2 \in O_F^2 \). A factor is defined to be exact if \( 1 \) state correspondence pairs for all states in \( O_F^1 \) to states in \( O_F^2 \) can be found such that no state appears in more than one correspondence pair and \( 2 \) for each internal edge in \( O_F^1 \), \( e_1 \), if \( \text{e}_1 \rightarrow \text{input} \cap \text{e}_2 \rightarrow \text{input} \) \( \neq \phi \) for any \( e_2 \) in \( O_F^2 \), \( (e_1 \rightarrow \text{fanout}, e_2 \rightarrow \text{fanout}) \) and \( (e_1 \rightarrow \text{fanin}, e_2 \rightarrow \text{fanin}) \) are state correspondence pairs. The definition of an exact factor can be extended for \( N_R > 2 \).

An ideal factor with \( N_R \) occurrences, each with \( N_E(i) + N_I(i) + 1 \) states, is an exact factor with \( N_E(i) \) entry states, \( N_I(i) \) internal states and a single exit state.

If two edges in a State Transition Graph can be represented by the same product term in an encoded and minimized two-level implementation, these two edges are said to be mergeable under that encoding.

3 The Global Strategy

In this section, we will describe the global strategy used in integrating factorization algorithms with state assignment techniques. An illustrative example will be presented and some theoretical results given.

The basic idea in our approach is to identify factoring states in a machine, and rather than performing a factorization, separately encode the states in the factored and factoring submachines. By separate encoding we mean that each submachine is encoded using a different set of bits. The strategy is described in detail below.

1. Given an original machine with \( N_S \) states, \( N_R \) disjoint sets of states, each with cardinality \( N_F \) are selected. These selected states represent occurrences of a factor and the selection of these sets is accomplished using factorization techniques that will be described in Sections 4, 5 and 6.

2. We perform two separate encoding steps using standard state assignment techniques. The unselected and selected states are encoded separately, using different fields of \( N_I \) and \( N_{I2} \) bits respectively.

3. The \( N_F \) states in each occurrence of \( F \), \( O_F \), are encoded using the second field of \( N_{I2} \geq \log(N_F) \) bits. Each occurrence is coded in exactly the same way - corresponding states are given the same code.

4. The unselected states are encoded using the first field of \( N_{I1} \) bits. The states in the \( N_R \) occurrences that were given the same code in Step 3 are differentiated using the first field. Each occurrence is given a code distinct from any of the unselected states.
The second field for the unselected states is as yet unspecified. This field can be arbitrarily chosen. However, it is beneficial to uniformly code this field for each of the unselected states with a code used already in the factor occurrences.

In Figure 1, a machine with 10 states is shown. A factor with 2 occurrences, each with 3 states, namely, (s4, s5, s6) and (s7, s8, s9) is selected. The factor is ideal with a single entry state in each occurrence (s4, s7), a single internal state (s5, s8) and a single exit state (s6, s9). After factorization, the unselected and selected states are one-hot encoded separately as shown in Figure 2. We have, in our example, 4 unselected states, namely, s1, s2, s3 and s10. This means we have 6 bits in the first field and 3 bits in the second field. The second field code for the unselected states was chosen to coincide with the code assigned to the exit states in this case. Indeed, as we will show in Theorem 3.2, this ensures that the factorization is maximally exploited. Of course, instead of one-hot coding the two fields, an encoding of shorter length can be constructed. One can use state assignment programs like KISS [4] and MUSTANG [2] to perform Steps 3 and 4, depending on whether two-level or multi-level logic implementations are targeted.

We now state and prove a theorem that relate the numbers of product terms after one-hot encoding and two-level optimization in the original and factored machines.

**Lemma 3.1**: Two edges that fan out to different next states in a machine are not mergeable under a one-hot encoding.

**Theorem 3.2**: Given a machine $M$, let the number of product terms in a one-hot coded and logic minimized two-level implementation be $P_0$. If an ideal factor $F \in M$ is extracted, then the number of product terms obtained by one-hot coding the factored and factoring machines separately, $P_1$, is related to $P_0$ by the following inequality.

$$P_0 \geq P_1 + \sum_{i=1}^{N_F-1} (|e_m(i)| - 1) - 1$$

where $|e_m(i)|$ is the number of product terms obtained by one-hot encoding and minimizing the $e(i)$ internal edges in each $O_F^i$. The number of encoding bits used after factorization will be $(N_R - 1) \times (N_F - 1) - 1$ less than for the original machine.

**Proof**: We first show that ideal factorization and one-hot encoding do not prevent the merging of edges that are mergeable when one-hot coding the original machine. Due to factorization, the edges in the different occurrences of the factor cannot merge. For example, $e_1 \in e(i)$ cannot merge with $e_2 \in e(j)$ if $i \neq j$. But $e_1$ and $e_2$ cannot merge in the one-hot coded original machine as well because they fan out to different next states.

Also, edges in the factored and factoring machines cannot merge. It is conceivable that a $e_p \in fin(i)$ could merge with $e_q \in e(i)$ in the original machine. However, since $F$ is ideal and only has entry, internal and exit states, it means that no $e_p \in e(i)$ fans into the entry states and $e_q \in fin(i)$ can only fan into some entry state. Therefore, these edges could not have merged in the original machine either.

Similarly, some $e_p \in fout(i)$ could merge with $e_w \in EXT$ in the original machine, if they fan into the same state. The states outside $F$ are given second field codes corresponding the code of the exit state in each $O_F^i$ (Step 5 above). Thus, the exit state's complete code differs only in the first field from the codes of states outside $F$. This means that fanout edges from the exit state $fout(i)$ can merge with external edges $EXT$ in the factored machine also (by merging the first field in the present state, since they fan into the same next state).

We now show that the internal edges $e(i) \in O_F^i$ can be coalesced, unlike in the original machine. For the original machine we will have $\sum_{i=1}^{N_R} |e_m(i)|$ product terms corresponding to these edges, after one-hot coding and optimization. The state field is split in two parts in the factored machine, $fn_1$ and $fn_2$. In the worst case, in the next state logic, these two fields are realized separately with no sharing whatsoever. It is easy to see that $|e_m(i)|$ product terms suffice to realize $fn_2$, since the $e(i)$ are identical and the fanout states are targeted. (Step 5 above). Thus, the exit state's complete code.

Figure 1: Factorization prior to Encoding

Figure 2: State Assignment After Factorization
have been coded the same in \( f_n \). The \( \epsilon_m(i) \) realize the primary outputs as well. The \( f_{n1} \) field can be realized even more simply. For each \( O_P \) we have \( e(i) \) edges with the same next state field in \( f_{n1} \), corresponding to the code assigned to the \( O_P \) (Step 4 above). Thus, these edges can be merged into a single product term with a don't care primary input vector. The states in the product term correspond to all the entry and internal states in the \( O_P \). Therefore, the difference in the number of resulting product terms in the factored machine is at least:

\[
\sum_{i=1}^{N} |\epsilon_m(i)| - |\epsilon_m(N_R)| - N_R = \sum_{i=1}^{N} (|\epsilon_m(i)| - 1) - 1
\]

less than the original machine.

The number of encoding bits used in the original machine is \( N_5 \). In the factored machine, we need \( N_5 - N_R \times N_F + N_R \) bits for the first field and \( N_F \) for the second. The reduction in bits is therefore \((N_R-1) \times (N_F - 1) - 1\). Q.E.D.

State assignment techniques do not perform significantly better than one-hot encoding in terms of the number of product terms in the final PLA. KISS guarantees that its result is at least as small as a one-hot coded result, and in general produces results with product terms slightly better than a one-hot coded implementation. NOVA [8], a variation on KISS, produces implementations with generally greater product terms than KISS or one-hot encoding, but saves on the number of encoding bits used.

Thus, Theorem 3.1 is quite a strong result, since it provides a deterministic estimate of the number of product terms in the final implementation. The upper bound on the number of product terms resulting from encoding an ideally factorized machine using a KISS-style algorithm is always lower than using the KISS-style algorithm on the original machine. This upper bound is found using multiple-valued minimization [4] and as mentioned earlier, is a tight bound — results significantly better than this are seldom obtained. The larger the ideal factor (in terms of number of states or number of occurrences), the greater will be the gains in using the strategy described. Even extracting small ideal factors will produce better results. The smallest possible ideal factor is one with 2 states and 2 occurrences shown in Figure 3. The factor has one entry state and one exit state. It is highly probable that at least one of these factors will exist in a large machine.

The strategy presented can be generalized to multiple factorization. Several disjoint ideal factors may exist in a machine, that can be simultaneously factored to produce cumulative gains in the number of product terms. If the factors are not disjoint, then choices have to be made as to what factors to extract at the expense of losing others.

**Theorem 3.3** : If \( N \) ideal factors, \( F_1, F_2, ..., F_N \), are extracted which contain disjoint sets of states when performing a one-hot coding after factorization over one-hot coding the original machine is

\[
G = \sum_{j=1}^{N} g_j
\]

where \( g_j \) is the gain corresponding to extracting \( F_j \) alone.

**Proof** : We now have \( N + 1 \) fields that are coded separately, corresponding to the \( N \) sets of states in each factor and the \( N + 1 \)-th corresponding to the unselected states. By the arguments of Theorem 3.2, none of the \( f_{in}(j,i) \) can merge with \( e(j,i) \) in the original or factored machines for any \( F_j \). The one-hot coding on the factored machine is performed as follows. Every occurrence \( O_P \) has a distinct \( N + 1 \)-th field. Each of the states (including the exit state) in the \( F_j \) and the unselected states are given a \( k \)-th field code equal to the exit state's code of \( F_k \). Any \( fout(j,i) \) fans out of an exit state and the code of this state differs from the unselected states in the \( N + 1 \)-th field alone, implying that \( fout(j,i) \) and \( EXT \) are mergeable in the factored machine, if they were mergeable in the original machine. Similarly, the codes of the exit states in each occurrence of the factor also differ only in the \( N + 1 \)-th field alone, and hence if \( fout(j,i) \) and \( fout(k,i) \) could merge in the original machine, they are mergeable in the factored machine.

The gain due to extracting each factor \( F_j \) is due to the merging of the \( e(j,i) \), which are all disjoint and there could have been no merging across the \( e(j,i) \) in the original machine. Therefore the cumulative gain is

\[
G = \sum_{j=1}^{N} g_j
\]

Q.E.D.

So far, we have dealt with the two-level case. The problem of state assignment for the multi-level case corresponds to finding an encoding that minimizes the number of literals in the encoded and optimized machine. While we cannot deterministically predict the effects of multi-level logic optimization, we can relate the number of literals in the original and factored machines after one-hot encoding and two-level logic minimization, but prior to multi-level optimization.

**Theorem 3.4** : Given a machine \( M \), let the number of literals in a one-hot coded and logic minimized two-level implementation be \( L_0 \). If an ideal factor \( F \in M \) is extracted, then the number of literals obtained by one-hot coding the factored and factoring machines separately, \( L_1 \), is related to \( L_0 \) by the following inequality.

\[
L_0 \geq L_1 + \sum_{i=1}^{N-1} \text{LIT}(\epsilon_m(i)) - \frac{N}{2}
\]
where \( LIT(e_m(i)) \) corresponds to the number of literals obtained by one-hot encoding and minimizing the \( e(i) \) internal edges in each \( Op^i \) and \( EXT_m \) corresponds to the number of product terms after one-hot coding and minimizing the external edges \( EXT \).

Proof: The arguments of Theorem 3.2 corresponding to the merging of edges hold. However, the number of literals in the external edges \( EXT \) in the factored machine is higher because the number of literals in each state is 2, rather than 1. If \( n \) external edges merge, we have \( n + 1 \) literals in the present state space of the factored machine, as opposed to \( n \) in the original machine (It is \( n + 1 \) rather than \( 2n \) because the merging of edges implies one of the state fields in each of the edges is identical). Therefore, the number of extra literals is \( |EXT_m| \) (In the worst case of \( EXT \) not merging with any \( e(i) \)). The reduction in the number of literals is due to the merging of the \( e(i) \). The number of literals in these edges in the original machine is simply \( \sum_{i=1}^{N_R} LIT(e_m(i)) \). In the factored machine, we have only one set of these \( e(i) \) edges. The number of literals in this set is not the same, however, as in the original machine sets, since the number of literals in the present state space is higher (the number of literals in each state is 2, rather than 1). Consider the first next state field, \( fn_1 \). The first present state field will have \( N_R \) literals in each product term that realizes \( fn_1 \). The second present state field will have the same number of literals as in any \( e_m(i) \) in the lumped field of the original machine. Now consider the second next state field. \( N_R \) product terms suffice to realize this field. The number of literals in each of these product terms is simply \( N_R - 1 \), corresponding to the entry and internal states in \( F \), since the primary input field is a don't care. Therefore, the total number of literals in the factored machine is \( LIT(e_m(N_R)) + N_R \times |e_m(N_R)| + N_R \times (N_R - 1) \). The reduction in literals is then

\[
N_R - 1 \sum_{i=1}^{N_R} LIT(e_m(i)) - N_R \times |e_m(N_R)| - N_R \times (N_R - 1) - |EXT_m| \]

Q.E.D.

4 Ideal Factorization

In this section, we will present an algorithm for finding all ideal factors given a State Transition Graph description of a machine. The techniques of [3] that identify exact factors cannot be used, since they assumed the existence of a starting state in each occurrence from which all other states in the occurrence could be reached. We do not make this assumption here – an ideal factor may have multiple entry states and therefore no starting state.

These ideal factors are extracted prior to state assignment in order to improve the performance of state assignment algorithms. The factors extracted may overlap. Thus, not all factors can be extracted – extracting one factor may invalidate the other. In Sections 5 and 6, we discuss techniques for choosing the appropriate factor(s) to extract in order to maximize the reduction in logic complexity after encoding and optimization.

The procedure starts with all possible exit state sets and traces the fanin of the states so as to identify ideal factors.

1. First, all sets of states of cardinality equal to \( N_R \) whose fanin edges assert the same outputs, if driven by the same input combination, are found. These sets are stored in \( TR_1 \).
2. All sets of states of cardinality equal to \( N_R \) whose fanout edges assert the same outputs, if driven by the same input combination, regardless of what states they fan into, are found. These sets are stored in \( TR_2 \).
3. A (new) set of states \( S_E \in TR_1 \) is picked.
4. An attempt is made to construct an ideal factor(s) whose exit state set corresponds to \( S_E \). This is done by tracing the fanin of \( S_E \). The set of states that fan into each \( q_i \in S_E \), \( 1 \leq i \leq N_R \) are found, namely, \( fanin(q_i) \).
5. A check is made to see if the \( fanin(q_i) \) are in direct correspondence. Each correspondence set must belong to \( TR_2 \) and the edge relationships between \( q_i \) and \( fanin(q_i) \) and within the \( fanin(q_i) \) must be identical for all \( i \). If the check fails, go to Step 3.
6. \( F = q_i \cup fanin(q_i) \) may be an ideal factor, with \( q_i \), representing the exit state of occurrence \( Op^i \) and \( fanin(q_i) \) the entry states. If so, \( F \) is recorded as such.
7. The fanin of the \( fanin(q_i) \) are traced back. Given a state \( s_{ij} \in fanin(q_i) \), if the \( fanin(s_{ij}) \) are not in direct correspondence, then \( s_{ij} \) has to be an entry state. If \( s_{ij} \) is not an entry state (receives edges from \( s_{ik} \in fanin(q_i) \) or from \( q_i \)), go to 3.
8. For all states \( s_{ij} \), that do not have to be entry states, \( fanin(s_{ij}) \) are found. A check is made to see if \( s_{ij} \) can be an entry state for some ideal factor. Choices corresponding to treating each of these \( s_{ij} \) as an internal or entry state are exhaustively explored. If the \( s_{ij} \) is chosen to be an entry state, it is merely added to the current \( F \). Else if \( s_{ij} \) is internal, \( fanin(s_{ij}) \) is also added to \( F \). The various \( F's \) are checked for ideality and the ideal factors are recorded.
9. Steps 7-8 are repeated for the fanins of the \( fanin(s_{ij}) \) and so on.

The function \( fanin(arg) \) above returns all states such that an edge from \( arg \) to each of these states exists.

5 Identifying Near-Ideal Factors

While the number of ideal factors in a machine is typically small and all ideal factors can be enumerated, there may be large numbers of near-ideal factors. Extracting these factors does not provide the gain corresponding to Theorem 3.2 or Theorem 3.4, but could produce some reduction in the eventual number of product terms or literals. We therefore have to solve the problem of detecting non-ideal, but good, factors and estimating the gain in extracting them. If \( cy \) can estimate the gain of a non-ideal factor, then a search procedure similar to the one described in the previous section can be used to detect good factors, i.e. factors with large associated gains. This estimation
of gain is different for the two-level or multi-level cases and is described in the next section. In this section, modifications to the ideal factor search procedure to find near-ideal factors are described. These modifications are similar to those proposed in [3] to find near-exact factors.

1. Find similarity weights for all possible $N_R$ sets of states. These weights are found on the basis of I/O fanout and fanin relationships between each set of states, i.e. the number of input symbols for which edges fanning out of all states in the set have different outputs. A weight of zero would correspond to exactly similar states.

2. These sets are ordered in terms of increasing weights (decreasing similarity). Beginning from each initial exit state set, the fanins of the set are traced as in Section 4.

3. The fanin states from the initial set are found and added to the factor (Step 5 in Section 4). The gain is extracting the current factor is estimated (described in Section 6). If the gain is below a prescribed value, the search is terminated and a new iteration with a new initial set is begun. Else, the factor and its associated gain are recorded and fanin tracing continues.

Thus, given $N_R$, one can find non-ideal factors, estimate the gains in extracting them and select factors with estimated gain greater than a prescribed value. This value is a function of the number of states in the non-ideal factor whose gain is being estimated. Larger factors, i.e. factors with more states, require a greater estimated gain in order to be recorded and in order for the search to continue. This is done because of the estimation of gain for non-ideal factors is approximate.

6 Implementation-Specific Factorization Techniques

6.1 Targeting Two-Level Logic

Given a machine and all the ideal factors in the machine, an appropriate set of factors has to be selected so as to maximize the cumulative gain in extracting these factors. The factors may be disjoint or may overlap. In the latter case, extracting one factor may invalidate the other. Thus, a step that selects the largest (maximum gain), non-overlapping set of factors has to be performed prior to state encoding. However, since the number of ideal factors is generally not very large, this step can be performed optimally via exhaustive search.

The issue extracting of non-ideal, but good factors is important. Since two-level implementations are quite constrained, even a small non-ideality in a factor can result in negligible gain in the number of product terms when extracting the factor. Hence, ideal factors are always extracted if they exist. A search can be performed on the ideally-factored machine, using the procedure described in the previous section, for good factors, or if no ideal factors exist, on the original machine. The gain of non-ideal factors in the two-level case is measured by

$$\sum_{i=1}^{N_R} |e_m(i)| - \left| \left( \bigcup_{i=1}^{N_R} e'(i) \right) \right|$$

The gain is measured by the number of product terms, by one-hot coding and minimizing $e(i)$ separately. $e'(i)$ corresponds to $e(i) \in O_F$ except that corresponding states in each $O_F$ are given the same codes (as when factoring).

Near-ideal factors are selectively extracted based on their estimated maximum cumulative gain.

6.2 Targeting Multi-Level Logic

In the multi-level case, non-ideal, but good factors play a more important part. First, all ideal factors are found and their gains calculated. However, these factors are not immediately extracted as in the two-level case. Near-ideal factors with the largest estimated gain in literals are found using the methods of Section 5. The gain of ideal and non-ideal factors is measured by

$$\sum_{i=1}^{N_R} \text{LIT}(e_m(i)) - \text{LIT}(\bigcup_{i=1}^{N_R} e'(i))$$

where $e_m(i)$ and $e'(i)$ are the same as before. A selection of non-overlapping ideal and near-ideal factors is made so as to maximize the overall gain.

7 Results

In this section, we present some preliminary results using the factorization algorithms presented in the previous sections, prior to state assignment.

The statistics of benchmark examples from the MCNC 1987 Logic Synthesis Workshop and other sources are given in Table 1. The examples were first state minimized. In Table 1, the number of inputs (inp), outputs (out), states (sta) and the minimum number of bits (min-enc) required for encoding are given. In Table 2, comparisons are drawn against the state assignment program KISS. The number of encoding bits used (eb) and the number of product terms (prod) required are given for KISS. The results for factorization followed by a KISS-style algorithm are given under FACTORIZE. The CPU times required for factorization and state assignment were nominal in all cases. The number of occurrences of the extracted factor (occ) and the type of factor extracted (typ = NOI for non-ideal) are indicated for each example. As the results indicate, ideal (or close to ideal) factors exist in large machines and extracting them produces better results.

<table>
<thead>
<tr>
<th>Example</th>
<th>inp</th>
<th>out</th>
<th>sta</th>
<th>min-enc</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mod12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>planet</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
</tr>
<tr>
<td>sand</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>styx</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>scf</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>indus1</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>indus2</td>
<td>16</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>cont1</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>cont2</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1: State Machine Statistics
This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

8 Conclusions
We have presented state assignment techniques targeting two-level and multi-level logic implementations based on state machine factorization algorithms followed by state assignment algorithms. These techniques produce results that are superior to previous approaches to state assignment.

9 Acknowledgements
This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.