On Shift Register Realization of Sequential Circuits

Shunichi Toida

April 4, 1996

Abstract

Realizing a sequential machine with shift registers is an attractive approach to the design for testability. It requires little extra circuit for scan and since feedbacks are always to the first flip-flop of a shift register test generation is expected to be easier than that for circuits with unstructured feedbacks. Though the shift register realization of sequential machines was studied by several researchers in 60's, there is still no satisfactory way of telling whether or not a given sequential machine is realizable with a given number of shift registers and/or flip-flops. This research is an attempt to find a reasonable, if not optimum, way of realizing a sequential machine with shift registers.

What is reported in this paper is a simple necessary and sufficient condition for a given sequential circuit to be realizable with shift registers, an efficient algorithm to test whether or not a given sequential machine is realizable with shift registers, and a simple algorithm to find a coding of the states is also given.

1 Introduction

With the increasing degree of integration of logic circuits the automatic generation of tests for detecting faults in logic circuits is becoming increasingly more time consuming. Though for combinational logic circuits a number of good algorithms have been developed for detecting stuck-at faults and they seem to be performing satisfactorily[?][?], for other types of faults such as delay faults and bridging faults there is still a lot of work remaining. For sequential logic circuits the situation is even worse. Though there are a number of methods for test generation for sequential circuits such as (1) scan methods, (2) time domain expansion methods, (3) BIST, and (4) design for testability, none of them is satisfactory. For (1) the area overhead is too large, (2) is not fast enough, (3) generally does not give good fault coverage, and (4) has not been successful yet.

It has been observed, though no formal proof has been found, that for sequential circuits with many feedback loops between the memory elements the test generation tends to take a lot of time. Based on this observation Cheng and Agrawal[?] have recently proposed a partial scan method in which a fraction of the memory elements are selected for scan so that all the feedback loops are broken if they are removed. This makes test generation easier without incurring much area overhead. Along the same line they have also suggested a state assignment method which minimizes the number of feedbacks between the memory elements. In the light of these works the shift register realization of sequential circuits, which was studied by several researchers in the past [?][?] [?] [?] [?] ([?] also contains a list of other works), is very interesting and requires reexamination from the testing point of view. For if a sequential circuit is realized using a number of shift registers, then very little extra circuit is required to scan it. Also since the feedbacks are always to the first memory element of a shift register, the test generation will also be easier for shift registers compared with circuits with unstructured feedbacks.

In one of the earlier papers on shift register realization of sequential machines Liu[?] examined relationships between the successors of states and tried to develop an algorithm for realizing a sequential machine with shift registers of the same length. But his method is rather incomplete. Later Nichols[?] investigated relationships between the predecessors as well as successors of states and developed a method to find an optimum shift register realization for a given sequential machine. But his method is basically an exhaustive search. Martin's work[?] is a little different from Liu, Nichols and others who sought unique code word for each state. In his method a state can be assigned two or more code words. Thus in effect extra states are added so that the entire new machine becomes a shift register. This problem is known as the equivalent-state problem.

In this paper the shift register realization of sequential circuits has been investigated along the line of Liu's work[?] with the specific purpose of design for testability. The major results obtained are a simple necessary and sufficient condition for a sequential machine to be realizable as a multivalued shift register, and a simple fast heuristic algorithm to find out whether or not a given sequential machine can be realized with a shift register of length k based on the condition. Theoretical analysis was also done on the algorithm, which suggests that the algorithm produces the longest shift register in most cases.

In section **2.** the problem treated in this paper is defined and terminology used is presented. In section **3.** a necessary and sufficient condition for a sequential machine to be realizable with a shift register of a given length is formulated and proved. Based on this condition a method to find a coding of the states for the machine is presented. In section **4.** structures which prevent k from becoming large are examined. Based on the results of the analysis a simple fast heuristic algorithm is suggested.

2 Preliminaries

A *partition* P of a set S is a collection of disjoint subsets of S such that the union of the subsets is equal to S. The subsets are called *blocks* of partition P. A partition of a set S is denoted by $\underline{0}$ if all of its blocks are a singleton, and by $\underline{1}$ if it has exactly one block. Let P and Q be partitions of a set S. Then the *sum* of P and Q, denoted by $P + Q$, is defined as the partition such that every pair of elements x and y of S are in the same block of $P + Q$ iff there is a sequence $x_0 = x$, x_1, ..., $x_k = y$, of elements of S, where for all $i < k$, either x_i and x_{i+1} are in the same block of P or Q. The *product* of P and Q, denoted by $P * Q$, is defined as the partition such that every pair of elements x and y of S are in the same block of $P * Q$ iff the are in the same block in P and also in Q.

???? Refinement P ¿ Q ????

A sequence of arcs of a directed graph(digraph) with their endpoints is called a *walk* from the first vertex to the last vertex, if the head of an arc is the tail of the next arc in the sequence for every arc. If no arc appears more than once in a walk, the walk is called a *trail*. If no vertices appear more than once in a trail, the trail is called a *path*. A directed graph is *strongly connected*, if there is a path between any pair of vertices in both directions. A digraph G is said to be *realizable with m shift registers* if every vertex of G can be labeled with a concatenation of m strings of possibly different lengths of symbols 0 and 1, called *factors*, in the following way:
There is an arc from vertex u to vertex v of G if and only if the label of v is obtained by shifting each factor of the label of u to the left one position and

inserting 0 or 1 into the rightmost position of each resulting factor. This lable for a vertex of G is called a *code word* for the vertex and the labeling is called a *coding* for G. The factors in a code word are assumed to be ordered in some order.
A graph with n vertices and with arcs in either direction between every pair of the vertices including self-loops is denoted by CD_n. In this paper unless otherwise specified a simple *graph* means a strongly connected graph. Hence it is assumed that a sequential circuit does not have transient states.

Definition 1 *Given a graph G, relations C and R are defined on the vertex set V of G as follows:*
For every u and $v \in V$, $(u, v) \in C$ iff there is a vertex $w \in V$ such that arcs (u, w) and (v, w) exist in G.
Let T_c denote the transitive closure of C.
Similarly for every u and $v \in V$, $(u, v) \in R$ iff there is a vertex $w \in V$ such that arcs (w, u) and (w, v) exist in G.
Let T_r denote the transitive closure of R.

The following lemma can easily be seen from the definition of T_c and T_r.

Lemma 1 *T_c and T_r are an equivalence relation on V.*

Let us denote by P_c and P_r the partitions induced by T_c and T_r, respectively.

Let S be the state set of a sequerntial machine with the transition function δ. Let P and Q be partitions of S with m blocks. P is said to *co-map* into Q, written $[P, Q]$, iff the blocks of P and Q are ordered so that $\delta(p_i) \subseteq q_i$ and $\delta^{-1}(q_i) \subseteq p_i$ for all $i \leq m$. Here p_i and q_i denote a block of P and Q, resperctively, and $\delta(p)$ for a block p of P or Q is the union of the images of p under δ over the set I of all input symbols, that is $\delta(p) = \{x : \exists s \in S \exists i \in I \delta(s, i) = x\}$.
Terms to be defined for this chapter:

??????

3 State of the Art

The realization of sequential machines with shift registers has been studied extensively by many researchers since early '60s. Virtually all the works examine the effects of state transitions on states as opposed to, for example, studying structures of state transition graph. These works can further be classified into

two categories. In one group, mainly partitions of a state set induced by the state transitions are investigated. Each state is given a single code word which is determined by the intersection of various partitions. For example Roome and Torng [?] and Nichols [?] belong to this group. In the works of the second group, equivalent states are allowed, that is a state can be assigned multiple code words. For example, Martin [?] and Friedman [?] belong to this group.
Below some of the works relevant to this current research are briefly summarized.

Hartmanis and Stearns[?, ?, ?, ?, ?] study loop-free structure of sequential machines using partitions of state sets. A machine M_i is a predecessor of a machine M_j iff the output of M_i is directly used as an input to M_j. If there is no loop by this predecessor relation, then it is called loop-free.
First the substitution property for a partition of the state set of a sequential machine is defined as follows:
A partition P on the set of states of the machine
$M = (S, I, \delta)$,
where S is the set of states, I is the set of input symbols, and δ is the transition function (outputs are ignored in this report), is said to have the *substitution property*(S.P.) iff
for any pair of states s and t of S, if s and t are in the same block of P, then $\delta(s, a)$ and $\delta(t, a)$ are also in the same block of P for all a in I.

(They show that the set of all partitions with the substitution property on the set of states of a sequential machine is closed under sum and product operations, and it forms a lattice under the natural ordering.)

They show that for each set of partitions which have the substitution property and whose product is $\underline{0}$ there is a loop-free realization of the sequential machine.
All the S.P. partitions of a sequential machine can be obtained by the following two step procedure:
(1) For every pair of states s and t, compute the smallest S.P. partition, $P^m_{s,t}$, which puts the pair into one block.
(2) Find all possible sums of the $P^m_{s,t}$. These sums constitute all the S.P. partitions.

Based on these results Friedman [?] showed, among other things, that an arbitrary sequential machine can be realized with at most one feedback with possible modifications to the given circuit. He introduces an "implication graph", and shows that if there is no cycle in the implication graph of a machine, then it is realizable with one feedback. He also presented an algorithm to destroy cycles in an implication graph by using equivalent states. This approach requires in general a very large number of equivalent states to realize a sequential machine with shift registers.

Later Haring [?] shows a variation of Friedman's realization using universal sequential machines.
Arnold et al. [?] propose an iterative realization of a sequential machine using AND-OR-DELAY and AND-EXOR-DELAY modules based on the results of Friedman [?]. This was later improved by Chao and Loomis [?], and Saluja [?]. Gray et al [?], and Prasad and Gray [?] discuss fault diagnosis of these realizations. These realizations are, however, mainly for circuits with one binary input and they use a large number of gates even for those circuits. For example Arnold et al [?] give an upper bound of n^4 for the number of modules, where n is the number of states.

Liu [?] defines a k-th order automaton to be an automaton whose next state depends essentially on its most recent k states as well as its present input, and gives a necessary condition for an automaton to be equivalent to some k-th order automaton and a sufficient condition to be equivalent to some third-order automaton. His results essentially are as follows:
First homomorphs are constructed iteratively one after another by coalescing the states in each block of P_r of each homomorph.
An automaton is equivalent to a k-th order automaton if and only if k homomorphs can be constructed that way and the states in each block of P_r for each homomorph map to different blocks of P_r. The memory elements in his results are in general of more than two values. Hence shift registers are not necessarily binary. Furthermore k is only up to three. Later Johnson and O'Keefe [?] completed what Liu left off.

Nichols [?] realizes sequential machines with the least number of shift registers. A single code word is assigned to a state. The search for optimum realization is essentially exhaustive. Nichols finds a shift register realization by first
(1) generating all SRP's, then
(2) selecting from this list the least number of SRP's that meet in $\underline{0}$.

To generate all SRP's proceed as follows:
(1) Merge the blocks of P_c in all possible ways to form binary partitions.
(Note that a binary partition thus obtained is always an SRP).
(2) For each partition so generated, form the SRP with which it has the co-mapping property. Denote the set of SRP's thus obtained by $\{P^1\}$.
(3) **repeat** the following:
    Obtain $\{P^{L+1}\}$ from $\{P^L\}$ by collecting $P_a^L * P_b^L$,
    where P_a^L, $P_b^L \in \{P^L\}$, and P_a^L has a co-mapping property with P_b^L.
  until $\{P^L\}$ is found in which no partition P satisfies $P_c \leq P$.

Roome and Torng developed the concept of P-tree and utilizes it to narrow the search space for finding a set of desirable shift registers.

A P-tree is a hierarchy of chains of partition of different lengths. It starts at P_c, and P_r, and goes up to $\underline{1}$. The length of chain increases by 1 from 2 everytime the level is increased. The i-th partition at level k is denoted by $P_{k,i}$, and it is obtained by taking the sum of $P_{k-1,i-1}$ and $P_{k-1,i}$.

They proved that if P_i is the i-th partition of a sequence of binary partitions of length k, then $P_{k,i}$ is a refinment of P_i ($P_i \geq P_{k,i}$), and conversely.
Hence a sequence of binary shift register partitions of length k (k-CC in their notation) is started by a partition obtained by merging blocks of $P_{k,1}$ into two blocks.
Roome and Torng go further into an elaborate scheme of pruning. However, there are cases when there is no P-tree. In such a case this method can not be used.

Two types of shift register realizations are considered by Martin: realizations allowing equivalent states i.e. multiple code words for a state, and realizations without equivalent states. But the emphasis is on the former.

Before describing Martin's results a few terminologies are introduced.
A *memory span test* is the following procedure:
(1) starting with the partition $\underline{1}$ of the state set,
(2) apply the transition function to the states of the partition of the state set, and group the successor states according to the input value to obtain a new partition.
(3) repeat (2) until either $\underline{0}$ is reached or a block of at least two states, which are not separated anywhere else, starts appearing.
A *finite memory span test* is a memory span test of finite steps. The sequence of input symbols used to reach a state from $\underline{1}$ in memory span test is the code word for the state. If a state is reached in more than one way, then it has equivalent states.

A *transform* of a state transition table(STT for short) is an STT obtained by any of the following operations:
(1) If a state appears more than once in a row of the STT, replace any number of them (but not all) with don't cares.
(2) Any finite number of states equivalent to any state of the sequential machine (SM) may be incorporated into the STT.
(3) Arbitrary permutations of the order of the successors of any state are allowed.
(4) A column of don't cares may be appended or deleted.

A *finite memory transform* is a transform which has a finite memory span test.

Pair-graph: It has one node for each unordered pair of states. A node N_1 is connected to nodes, states, or don't-cares by an arrow iff the states in N_1 map to the nodes, states, or don't-cares under a permutation of columns of STG. A pair-graph is called **complete** iff there are no cycles of pairs.
An SM has in general more than one pair-graph.

$\underline{SM}$ is an SM in which each nontransient state has two distinct successors.
$\overline{SM}$ is a two-column SM in which at least one nontransient state has less than two distinct successors.

Martin's results are mostly on $\underline{SM}$ and $\overline{SM}$, and equivalent states are allowed.

Some of the major results of Martin are listed below.

An SM without inaccessible states is L-level (i.e.L input values) feedback shift register realizable iff it has an L-column finite memory transform.

If an $\underline{SM}$ without transient states is binary shift register realizable(BFSRR), then it has a complete pair-graph.

He gives a $(n-1)$ step procedure to test an n state $\underline{SM}$ for BFSRR.
An $\overline{SM}$ which has a two-column, finite memory transform is BFSRR regardless of how many transient states it might have.

An $\underline{SM}$ which has a two-column, finite memory transform is not BFSRR, if it has transient states, but it can be realized with a BFSR and a one-bit delay in parallel.

The next two results do not use equivalent states.

An SM is FSRR iff $m^* M(m^*) = \underline{0}$. $m^* = P_r$ and $M(m^*) = P_c$.

An NSC for bidirectional FSRR is also given.
But it essentially tries all possible combinations of direction for the edges.
As he also notes, he does not give any efficient algorithm to find a finite memory transform for a given SM. Thus except for $\underline{SM}$ without transient states, the search for BFSRR still requires a large amount of time in general.

Su and Yau [?] discuss unitary realization of sequential machines. A realization is called unitary if all the code words assigned to a state have ℓ identical bits, where ℓ is the number of shift registers in the realization.
Thus multiple code words are allowed and the least number of shift registers are sought.

For multiple unitary SR realization, unitarily realizable partitions $P_1, P_2, ..., P_k$ such that $\Pi_{i=1}^{k} P_i = \underline{0}$ are exhaustively searched for. Also it is very difficult to control the number of equivalent states.

Since a sequential machine can not always be realized with shift registers unless a very large number of equivalent states are used, Agrawal and Cheng [?] suggest to embed a test machine into a given sequential machine which can be controlled easily. They explain this idea using shift register as an example of test machine. If one seeks a smallest shift register to embed in a given sequential machine, this problem amounts to realizing a subset of state set of the given machine with a shift register. They, however, do not propose any algorithm to identify shift register(s) to be embedded in a sequential machine.
Following the idea of Agrawal and Cheng [?], Kanjilal et al [?] developed an algorithm to find an embedding of a machine with the state transition graph of a shift register into a machine to be tested. It is polynomial time if the object machine is completely specified. Otherwise it is not necessarily polynomial time. They also discuss the use of an extra input line for embedding. Further they investigate embedding of test machines in interconnected machines.

4 Realizability Condition

Not all sequential machines are realizable with shift registers except the trivial realization of using only shift registers of length 1. In this chapter first a necessary and sufficient condition is proved for a sequential machine to be realizable with shift registers at least one of which is of length 2 or larger. Then the results of a computational experiment are presented which was conducted to estimate how much of a given sequential machine can be realized by shift registers using this necessary and sufficient condition.

In this chapter it is assumed that a sequential machine does not have any transient states and that a transition graph is strongly connected.

4.1 Realizability Condition

Theorem 1 *If a digraph $G(V, A)$ is realizable with shift registers at least one of which is of length 2 or larger (simply realizable with shift registers hereafter), then $P_c * P_r = \underline{0}$.*

Proof: This is proven by proof by contradiction.
Suppose that G is realized with shift registers and that $P_c * P_r \neq \underline{0}$.
Then there are vertices x and y of V, a block B_c of P_c and a block B_r of P_r such that both x and y are in B_c and B_r.

Claim: Let $X_{11}...X_{1k_1}X_{21}...X_{2k_2}...X_{d1}...X_{dk_d}$, and $Y_{11}...Y_{1k_1}Y_{21}...Y_{2k_2}...Y_{d1}...Y_{dk_d}$ be the code word for vertices x and y, respectively, in the shift register realization of G, where d is the number of shift registers and k_i is the length of the i-th shift register, and X_{ij} and Y_{ij} are symbols from the alphabet.

Then for all i and j, $X_{ij} = Y_{ij}$.

Proof of Claim: Assume that code words shift left when an arc is traversed. Since x and y are in B_c and B_r, assume first that there are vertices s and t such that arcs $(s,x),(s,y),(x,t)$, and (y,t) exist in G. Since x and y go to the same vertex t, $X_{ij} = Y_{ij}$ for all i, $1 \le i \le d$ and for all corresponding j, $2 \le j \le k_i + 1$. Similarly since s goes to x and y, $X_{ij} = Y_{ij}$ for all i, $1 \le i \le d$ and for all corresponding j, $1 \le j \le k_i - 1$.
Hence $X_{ij} = Y_{ij}$ for all i and j.

In general, by simple transitivity of equality, $X_{ij} = Y_{ij}$ holds for all i and j, if x and y are in B_c and B_r. ♣

If $P_c * P_r \neq \underline{0}$, then there are vertices x and y which are in the same block in P_c and in P_r. Hence by the claim x and y have the same code word. This contradicts that G is realized with shift registers. ♣

The converse of Theorem1 is trivially true, if a multivalued shift register is used. It can, however, also be proven true, when shift registers are restricted to binary ones.

Theorem 2 *If $P_c * P_r = \underline{0}$ for a graph $G(V,A)$, then G is realizable with binary shift registers.*

Proof: Two cases are considered:the case when there is a block of size at least 2 in P_c hence also in P_r, and the case when all the blocks are of size 1. For each case, binary partitions P_{c1} and P_{r1} are going to be found which satisfy $P_{c1} > P_c$ and $P_{r1} > P_r$.
[Case 1] There is a block, call it B_{c1}, of size at least 2 in P_c.
Let x and y be two vertices in B_{c1}. Then since $P_c * P_r = \underline{0}$, x and y must be in different blocks of P_r. Hence there are at least two blocks in P_r, hence also in P_c. Let B_{r1} and B_{r2} be the two blocks of P_r which contain x and y, respectively. Then construct a binary partition from P_r by taking union of blocks of P_r so that B_{r1} and B_{r2} are put into different blocks. Call this binary partition BP_r. Corresponding to BP_r(by δ^{-1}) there is a binary partition of P_c. Call it BP_c. Then $BP_c * BP_r$ has at least three blocks. Hence the shift register of length 2 which is determined by this set of binary partitions partitions the set of vertices into at least three sets. Clearly the rest of the vertices, which are not distinguished by this shift register, can be distinguished (at the worst) by

a series of shift registers of length 1. Hence the theorem holds in this case.
[Case 2] All blocks of P_c are of size 1.
In this case there is no vertex which is mapped onto itself by δ, because G is assumed to be strongly connected. Also it can be assumed that there are at least three vertices in G (for otherwise only one bit is needed to distinguish the vertices). Then three blocks, say $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, in P_c and a binary partition, call it BP_c, of P_c can always be found such that x_1 and x_2 are in one block and x_3 in the other in BP_c, and in the corresponding partition of P_r, call it BP_r, x_1 and x_2 are in different blocks. Then $BP_c * BP_r$ has at least three blocks. Hence as in Case 1, G is also realizable with binary shift registers in this case.
♣

Note: A binary partition of length 2 or larger can always be found after the first one as long as the refinement of P_c (hence P_r) falls into one of the two cases of this proof.

4.2 Testing for Realizability

A simple preliminary computational experiment has been conducted to have a rough estimate on the possibility of satisfying the shift register realizability condition. The results show that none of the MCNC benchmark circuits (except the shift register) satisfies the condition, and further that none of 100 randomly generated state transition diagrams satisfies that. There are too many transitions for most sequential machines to be shift register realizable. The next question, then, is how many of the transitions of a sequential machine can be realized by shift registers. This section addresses that question.

The realizability condition $P_c * P_r = \underline{0}$ is represented using a bipartite graph, and the problem of finding the minimum number of transitions to be removed to satisfy the condition is solved. The results of computational experiments on some of the MCNC benchmark circuits are also presented.

To identify the transitions to be removed, the following bipartite graph is utilized. For a given digraph $D(S, A)$ with a vertex set S and an arc set A, a bipartite graph $G = (V, E)$ is constructed as follows: $V = V_1 \cup V_2$, $V_i = \{< x, i >| \, x \in S\}$ for $i = 1, 2$, and $E \subset V_1 \times V_2$ such that an edge $(< x, 1 >, < y, 2 >)$ is in E iff there is an arc from x to y in A. G is called the bipartite graph corresponding to D. Using this graph G, the necessary and sufficient condition $P_c * P_r = \underline{0}$ can be reformulated as follows.

Theorem 3 *$P_c * P_r = \underline{0}$ iff for every s_1 and $s_2 \in S$, either vertices $< s_1, 1 >$ and $< s_2, 1 >$ are not connected in G, or vertices $< s_1, 2 >$ and $< s_2, 2 >$ are not connected in G.*

Proof: Let B_{c1} and B_{c2} be blocks of P_c and let B_{r1} and B_{r2} be blocks of P_r.

Then we can claim the following.
Claim: The vertices $< s_i, 1 >$ of V_1 with $s_i \in B_{c1}$ and the vertices $< s_j, 1 >$ of V_1 with $s_j \in B_{c2}$ are in the same connected component of G iff B_{c1} and B_{c2} are identical. Similarly for the vertices of B_{r1} and B_{r2}.
Proof of Claim: Let s_i and s_j be two vertices of $D(S, A)$. If $(s_i,\ s_j)$ is in the relation C of Definition 1 for $D(S, A)$, then there is a vertex s_k in S such that edges $(< s_i, 1 >, < s_k, 2 >)$, and $(< s_j, 1 >, < s_k, 2 >)$ are in G. The converse is also true. Hence $(s_i,\ s_j) \in T_c$ of Definition 1, iff there is a path between $< s_i, 1 >$ and $< s_j, 1 >$ in G. Hence the vertices $< s_i, 1 >$ of V_1 with $s_i \in B_{c1}$ and the vertices $< s_j, 1 >$ of V_1 with $s_j \in B_{c2}$ are in the same connected component of G iff B_{c1} and B_{c2} are identical. Similarly for the blocks of P_r. $\clubsuit$

Since $P_c * P_r = \underset{\sim}{0}$ iff for every pair of vertices x and y of S, x and y are in different blocks of P_c or P_r, from the claim the theorem follows immediately. $\clubsuit$

For a pair of vertices s_1 and s_2 of S, if either vertices $< s_1, 1 >$ and $< s_2, 1 >$ are not connected in G, or vertices $< s_1, 2 >$ and $< s_2, 2 >$ are not connected in G, then s_1 and s_2 are said to be **disconnected either in V_1 or in V_2**. When they are not disconnected in V_1 (or in V_2), they are **connected in** V_1 (or in V_2, respectively).

With this bipartite graph model, the problem of identifying non-shift register transitions becomes that of disconnecting every pair of vertices of the given digraph either in V_1 or in V_2.

Let us define the problem of identifying non-shift register transitions in a slightly more general setting as follows.

Definition 2 *Let $G(V, E)$ be an undirected bipartite graph. Let $\{V_1, V_2\}$ be the partition of the vertex set V and assume that $\mid V_1 \mid = \mid V_2 \mid$. Also assume that there is a bijection σ between V_1 and V_2 and that for every v_{i1} of V_1 $\sigma(v_{i1})$ is denoted by v_{i2}.*

Given a subset N of V_1, a pseudo N-cut *for N is a set of edges which, when removed from G, disconnect v_{i1} from v_{j1} in V_1 or $v_{i2}(= \sigma(v_{i1}))$ from $v_{j2}(= \sigma(v_{j1}))$ in V_2, for each pair of vertices v_{i1}, and v_{j1} of N.*

The problem of finding a smallest pseudo N-cut for a given subset N of V_1 is called a **modified N-cut problem**. Next this modified N-cut problem is going to be shown to be NP-complete.

Theorem 4 *The modified N-cut problem defined above is NP-complete.*

Proof: This is going to be proven by reducing the problem called N-cut problem to the modified N-cut problem. The N-cut problem is defined as follows and it is known to be NP-complete [?].
N-cut problem: Given an undirected graph $G(V, E)$ and a subset N of V, find a smallest set of edges which disconnect every pair of vertices in N when removed from G.

An instance of the N-cut problem can be transformed into an instance of the modified N-cut problem as follows: Let $G(V, E)$ be an undirected graph and let N be a subset of V of an instance of the N-cut problem for G. Let V_{11} be a copy of V. Also construct a set of vertices V_{12} by placing a vertex into it corresponding to each edge of E. Let V_{21} be a copy of V_{11} and let V_{22} be a copy of V_{12}. A bipartite graph G' is constructed with vertex set $V_1 \cup V_2$, where $V_i = V_{i1} \cup V_{i2}$, $i = 1, 2$, and with edge set $E' \subseteq V_1 \times V_2$ such that for every vertex u of V_{11}(and V_{21}) and every vertex v of V_{22}(and V_{12}, respectively), (u, v) is in E' iff the edge of E corresponding to v is incident to the vertex of V corresponding to u.

Then, as can be easily seen, the vertices of N are disconnected in G iff there is a pseudo N-cut for N in G'. Also since at most one of the edges incident to each vertex of V_{22} (i.e. an edge of G) needs to be removed for a cut in G', there is a one-to-one correspondence between the edges of E to be removed and the edges of E' to be removed. Hence a solution to the N-cut problem can be obtained by solving the modified N-cut problem. Hence the modified N-cut problem is also NP-complete. ♣

Note: Theorem 4 does not necessarily mean that the problem of finding a smallest set of edges is NP-complete which are to be removed to make a graph realizable with shift registers. For in the modified N-cut problem, a subset of vertices rather than the entire set of vertices are to be disconnected and that might make the problem harder. It is, however, expected that the modified N-cut problem is equally difficult when $N = V_1$.

4.3 Randomized Algorithm

The problem of finding a smallest set of edges to be removed to make a given digraph shift register realizable is most likely NP-complete, though it has not been proven. One may look for an efficient alogorithm to produce an exact solution. However, since the main purpose is to get a rough estimate of the percentage of transitions realizable with shift registers, and since the minimality requirement is not crucial (there is no real harm if a few more transitions are not covered by shift registers), here an approach is taken to find an approximate solution. In particular a randomized algorithm of [?] is adopted. The randomized algorithm is a modification of the simulated annealing method. Unlike the simulated annealing method, however, in the randomized algorithm, a favorable search direction may also not be selected with a certain probability, though the probability of not selecting a favorable direction is smaller than that of not selecting an unfavorable direction. This approach has been applied to the max clique problem and very good results have been reported[?].

In this section first the skeleton of the randomized algorithm is presented. Then its adaptation to the shift register realization is discussed.

The randomized algorithm in general form can be described as follows.

procedure Randomized_Algorithm

var1 current;
var2 candidate_solution;

inititalize; * Initialize **current** and **candidate_solution**. *\
while(not iterated sufficiently){
    for each neighbor i of **current**
        if $sigm(-\frac{\Delta En(i)+Rel(i)}{T}) > rand(0,1)${
            current := i;
            if candidate_solution is improved by moving to i
                update **candidate_solution**
        }
}

Here $En(i)$ is the energy function i.e. the objective function, $Rel(i)$ is a relaxation term which is used to avoid being trapped in a local optimum, T is a constant, which corresponds to the temperature in simulated annealing, $rand(0,1)$ is a random number between 0 and 1, and $sigm(x) = (1 + e^x)^{-1}$.

This Randomized_Algorithm can be applied to the problem of finding a minimum set of edges to be removed to make a given graph shift register realizable as follows:

Given a digraph $D(S, A)$ let $G(V, E)$ be the bipartite graph corresponding to D, and let $\{V_1, V_2\}$ be the bipartite partition of V. Order the edges of E in some order. Then a set of edges of E can be represented by an m-tuple of 0's and 1's, where $m = \mid E \mid$, and the i-th bit(say from left) of an m-tuple corresponds to the i-th edge. The energy function for an m-tuple x can be defined as follows:

Let $L(x)$ be the number of 1's in x, and let $C(x)$ be the number of pairs of vertices of S which are *connected in* V_1 and *in* V_2 when the edges of x are removed from G. Then the energy function $En(x)$ for x is defined as

$$En(x) = L(x) + wC(x),$$

where w is a weight whose value is determined depending on the importance of $L(x)$ and $C(x)$. To guarantee that an optimum x separates every pair of vertices of S, a large value must be chosen for w.

Randomized_Algorithm starts with an initial x, and generates and visits neighbors one by one with a certain probability. The i-th neighbor of x is generated by switching the value of i-th bit of x between 0 and 1.
The algorithm, which is called Mincut_Algorithm, is given in Fig. 1

Based on Randomized_Algorithm a computer program has been written to find a minimum set of edges to be removed to make a digraph shift register realizable, and computational experiments have been conducted on MCNC benchmark circuits. The results are summarized in Table 1

Mincut_Algorithm

Given a digraph $D(S,A)$, let $G(V,E)$ be the bipartite graph corresponding to D, and let $\{V_1, V_2\}$ be the bipartite partition of V. This algorithm finds a minimum set of edges to be removed from G to disconnect every pair of vertices of S either in V_1 or in V_2.

```
var1 x, oldx; \* x is the set of edges to be removed. *\
const Max_Iter; \* Max_Iter gives the number of iterations. *\

initialize x;
remove the edges of x from G;
find the components of the resultant graph, denoted by G − x;
compute En(x);
for (t = 1 to t = Max_Iter){
      for (i = 0 to i =| E | by 1){
            if (e_i is in x) \* e_i is the i-th edge. *\
                  if (e_i connects two components by itself){
                        oldx := x;
                        x := x - {e_i};
                        compute C(x);
                  }
            else{ \* e_i is not in x *\
                  find the component C_j to which e_i belongs;
                  depth first search C_j from an endpoint of e_i;
                  if (the other endpoint of e_i is not in C_j){
                        oldx := x;
                        x = x ∪ {e_i};
                        compute C(x);
                  }
            } \* end else *\
            compute En(x);
            if (sigm(−(ΔEn(i)+Rel(i))/T) ≤ rand(0,1))
                  x := oldx;
      } \* end for i *\
} \* end for t *\
```

Table 1 Computation Results

Name of Circuit	Number of States	Number of Transitions†	Transitions to be Removed	Length of Shift Registers
bbara	10	37	17	2-2-2
bbsse	16	42	13	2-2-2
bbtas	6	12	3	2-2
beecount	7	23	10	2-2
cse	16	55	25	2-2-2-2
dk14	7	27	11	2-2
dk15	4	12	5	2
dk16	27	102	25	2-2-2
dk17	8	23	7	2-2
dk27	7	13	1	2-2
dk512	15	30	4	2-2-2
donfile	24	96	39	2-2-2
ex1	20	73	30	2-2-2
ex4	14	18	0	4-2
ex6	8	31	13	2-2
keyb	19	46	15	2-2-2-2
lion	4	10	4	2
lion9	9	25	11	2-2
mc	4	8	2	2
modulo12	12	24	6	2-2-2
planet	48	71	6	2-2-2-2
s1	20	80	30	2-2-2
sand	32	90	36	2-2-2-2
shiftreg	8	16	0	3
styr	30	92	33	2-2-2-2
tav	4	4	0	2
train4	4	8	2	2
train11	11	25	8	2-2-2

† All the transitions from one state to another on different inputs are counted as one transition here.

5 Finding Shift Registers

In this chapter a method of finding shift registers is discussed for sequential circuits which are known to be realizable with shift registers. As can be seen from the State of the Art, there are but two methods that may be used here:Nichols'[?] and Roome and Torng's[?]. Both are an exhaustive search type algorithm and so neither one is applicable to large sequential machines. The problem of finding a smallest number of shift registers to realize a given sequential machine is, however, most likely NP-complete and an efficient algorithm is at best very hard to find. To cope with this difficulty, in this chapter the randomized algorithm is again applied. Since the randomized algorithm employed here relys on results of Roome and Torng[?], the relevant results are summarized first.

5.1 P Tree

The following theorems are all from Roome and Torng[?].

Theorem 5 *$[P_c, P_r]$, and if $[P, Q]$, then $P \geq P_c$ and $Q \geq P_r$.*

Theorem 6 *If $[P_a, Q_b]$, then for any $P \geq P_a$, there is a unique Q such that $Q \geq Q_b$ and $[P, Q]$. Also for any $Q \geq Q_b$, there is a unique P such that $P \geq P_a$ and $[P, Q]$.*

These results are also found in Nichols[?].

The following results show that there are certain key shift register partitions from which all the shift register partitions of any length can be generated. This is particularly well suited for the randomized algorithm.

Definition 3 *A set of k partitions $\{P_i\}$ is called a co-mapping chain of length k (denoted $k - CC$) iff $[P_i, P_{i+1}]$ for all $i < k$.*

Next a set of partitions called the *P tree* is defined constructively. The k partitions $\{P_{k,i}\}$, $1 \leq i \leq k$, which is a $k - CC$, is referred to as row of the tree.

Definition 4 *Let $P_{2,1} = P_c$ and $P_{2,2} = P_r$ form row 2.*
Row k is said to exist if $[P_{k,i}, P_{k,i+1}]$ for $1 \leq i < k$.
For $k \geq 3$, if row $k-1$ exists, form row k as follows:

> *$P_{k,i} = P_{k-1,i-1} + P_{k-1,i}$ for $2 \leq i \leq k-1$,*
> *construct $P_{k,1}$ so that $[P_{k,1}, P_{k,2}]$, and*
> *construct $P_{k,k}$ so that $[P_{k,k-1}, P_{k,k}]$.*

Note that row 2 always exists by Theorem ??.

Roome and Torng show that each row k generated in Definition ?? is a $k - CC$, for each k, $2 \leq k \leq$ Max k, that is

Theorem 7 *For any machine, and for all k, $[P_{k,i}, P_{k,i+1}]$ for $1 \leq i < k$.*

The P *tree* thus defined has the following very important property.

Theorem 8 *Let $\{P_i\}$ be a $k - CC$. Then $P_i \geq P_{k,i}$.*

Theorem 9 *Let $P \geq P_{k,j}$ for some $j \leq k$. Then there is a unique $k - CC$, $\{P_i\}$ such that $P_j = P$.*

Some of the relevant consequences of Roome and Torng's results are as follows.

1. If $P_c * P_r = 0$, then the number of blocks in $P_{k,1}$ decreases as k becomes large and $P_{k,1}$ eventually becomes $\underline{1}$.
2. The maximum length of shift registers that can be used to realize a given sequential machine is determined by the height of P *tree* i.e. the largest k for which $P_{k,1}$ is not $\underline{1}$.
3. Any binary partition P determines a binary shift register of length k, if $P \geq P_{k,1}$. Thus every binary partition of length k can be obtained by merging blocks of $P_{k,1}$ into two blocks.

These results are well suited for the randomized algorithm for finding shift registers to realize a given sequential machine. A number of different strategies can be considered for finding shift registers when a sequential machine is realizable with shift registers. Here shift registers are found starting with the largest k. For each k a shift register is sought by a randomized algorithm which, together with shift registers found so far, leaves the least number of states undistinguished. This is repeated until all states are distinguished from each other. The randomized algorithm examines different sets of blocks of $P_{k,1}$ to merge for each k to form a binary partition.

An outline of an algorithm utilizing $P_{k,1}$ is given below.

Algorithm Shift_Register_Realization

Given a digraph $D(S, A)$ which satisfies $P_c * P_r = \underline{1}$, this algorithm finds binary shift register partitions for S.

```
partition T, P;
int k;

find the P tree for D(S,A);     \* P_{k,1}'s are found here. *\
k := the largest k for D(S,A);
T := 1;
while (k ≥ 2 or T ≠ 0){
        if (| P_{k,1} |> 2)
                P := optimum_binary_partition(T, P_{k,1});
        else
                P := P_{k,1};
        store P and the current k; \* P is one of the desired shift registers. *\
        find the k − CC, P_1, ..., P_k, starting with P;
        T := T * Π_{i=1}^{k} P_i;
        k := k − 1;
}

procedure optimum_binary_partition(T, P_{k,1})

partition T, P_{k,1}, MinPart;
int En, NewEn, MinEn, n;
const Max_Iter, Rel, TMPR;

int function E(Q);
partition Q;
        E(Q) := the number of states in the largest block of partition Q;

int function H(Q);
partition Q;
        H(Q) := the number of blocks of partition Q;

n := the number of states;
let P_1 be a binary partition such that P_1 > P_{k,1};
find the k − CC, P_1, ..., P_k, starting with P_1;
Q := (Π_{i=1}^{k} P_i) * T;
En := nE(Q) + (n − H(Q)) + k;
MinEn := En;
MinPart := P_1;
```

```
for(t = 1 to t = Max_Iter){
    let P be a binary partition adjacent to P_1 such that P > P_{k,1};
    find the k - CC, P_1, ..., P_k, starting with P;
    Q := (Π_{i=1}^{k} P_i) * T;
    NewEn := nE(Q) + (n - H(Q)) + k;
    if (NewEn < MinEn){
        MinEn := NewEn;
        MinPart := P_1;
    }
    ΔEn := NewEn - En;
    if (sigm(-(ΔEn+Rel)/TMPR) > rand(0,1)){
        P_1 := P;
        En := NewEn;
    }
}
return MinPart;
```

A C program has been written to implement this algorithm and a computational experiment has been conducted using MCNC benchmark circuits. The results are given in Table 1.

Since these circuits are rather small, any general conclusion can not be drawn. However, shift registers obtained for these circuits are all short. One of the reasons for that is that the objective of the Min-Cut algorithm is to find the smallest number of transitions to be removed to make teh given machine realizable with shift registers, and that it does not attempt to make the length of the longest possible shift register large. For small circuits this may not make much difference. But for larger circuits, a new energy function is necessary which makes the largest k large.

6 Realizability Test II

In this section an algorithm is going to be presented for realizing a sequential machine with two shift registers of possibly different lengths. It is going to be shown that the algorithm always produces shift registers, if the sequential machine is realizable with two shift registers. It is also going to be proven that if the algorithm fails to produce shift registers, then the sequential machine is not realizable with two shift registers.

The algorithm finds shift registers by producing a coding of states such that for any transition the code word of the destination is obtained by shifting that of the source to the left by one bit. Each bit of this coding is found by successively obtaining homomorphs of the state diagram of the given sequential machine. Let us thus start with the definition of this homomorphism.

Definition 5 *A* k-th order homomorph G^k *of a graph* G *is defined recursively as follows:*

(1) $G^0 = G$.

(2) For $i = 1$ *to* k*, partition the vertex set* V^{i-1} *of* G^{i-1}*(denote the partition by* E^i*) so that*

(a) for any two vertices u *and* $v \in V^{i-1}$*, if* $(u, v) \in T_r$*, then* u *and* v *are in one block of the partition* E^i*,*

(b) for each block E^i_j *of partition* E^i *if* u *and* $v \in E^i_j$*, then the vertices to which* u *and* v *are connected in* G^{i-1} *are in different blocks of* E^i*, and*

(c) each block E^i_j *contains at most 4 vertices of* G^{i-1}*.*

If no such partition exists, then G^i does not exist.

(3) Construct G^i *by using a vertex for each block of the partition* E^i *and by connecting vertices* x *and* y *of* G^i *by an arc from* x *to* y *iff* (u, v) *is an arc of* G^{i-1}*, where* u *and* v *are vertices in the blocks of* E^{i-1} *represented by* x *and* y*, respectively.*

Obviously a homomorph is determined by a partition. Thus the homomorph G^i is called the *graph*(or *homomorph*) induced by the partition E^i.

Note that G^k is homomorphic to G, and if G is strongly connected then so is G^k.

For example let G be given by the following table.

Source	Destinations
0	0 , 1
1	2 , 3
2	4
3	6 , 7
4	1
5	2
6	4 , 5
7	7

Then R is $\{(0,1),(2,3),(4,5),(6,7),(1,0),(3,2),(5,4),(7,6)\}$.
Hence the partition E^1 based on T_r is $\{\ \{0,1\},\ \{2,3\},\ \{4,5\},\ \{6,7\}\}$.
Let $A_0 = \{0,1\}$, $A_1 = \{2,3\}$, $A_2 = \{4,5\}$, and $A_3 = \{6,7\}$. Then $V^1 = \{\ A_0,$ $A_1,\ A_2,\ A_3\}$, and the arc set of G^1 is $\{\ (A_0,A_0),\ (A_0,A_1),\ (A_1,A_2),\ (A_1,A_3),$ $(A_2,A_0),\ (A_2,A_1),\ (A_3,A_2),\ (A_3,A_3)\ \}$. Then the partition $E^2 = \{\ \{A_0,A_1\},$ $\{A_2,A_3\}\ \}$. Let $B_0 = (A_0,A_1)$, and $B_1 = (A_0,A_1)$. Then $V^2 = \{B_0,B_1\}$, and the arc set of G^2 is $\{\ (B_0,B_0),\ (B_0,B_1),\ (B_1,B_0),\ (B_1,B_1)\ \}$.

Using this homomorph, two shift registers which realize a given sequential circuit can be obtained by the following algorithm if it is realizable with two shift registers. If this algorithm fails to produce shift registers, more than two shift registers are necessary to realize the sequential circuit.

Algorithm SRR

Input: A state transition graph G of a sequential circuit
Output: A pair of factors for each vertex of G such that the code word of the destination state of each transition is obtained by shifting the factors of the code word of the source state to the left one position.

Step 1: Find the i-th order homomorph G^i, $i \geq 1$, of G successively and apply **Procedure Template Generation** for each G^i starting with G until one of the following two graphs is reached:
(a) CD_4 or its subgraph
(b) For every vertex of G^i, all of its outgoing arcs go to at most two different vertices(destinations). (Hence if the out-degree is greater than 2, then there are some parallel arcs in G^i and the vertices coded by a pair of parallel arcs get the same template). Let us denote this homomroph by G^K.

If (a) is the case, then go to Step 2, else if (b), then go to Step 3. If neither (a) nor (b), then G can not be realized with two shift registers.

Step 2: Assign $00, 01, 10$, and 11 arbitrarily to the four vertices of CD_4. These strings are interpreted as two 1-bit factors. Obtain two factors for each vertex of G by substituting each of these 1-bit factors into the templates for CD_4. The code word for a vertex of G is the concatenation of these two factors.

Step 3: At this point i.e. at the end of 1(b), one of the two factors for each vertex of G can be found as follows:
To each pair of vertices of G^K having an identical template, i.e. the destinations of parallel arcs, assign a 0 or 1 to distinguish them from each other. Assign all the other vertices a 0 or 1 arbitrarily. Then substitute it into the template of each vertex. The result is one of the two factors for the vertex.

To find the other factor for the vertices of G, go to 4.

Step 4: Merge each pair of parallel arcs, hence their destination vertices.
Then continue finding i-th order homomorph of G^K and templates but with at most two vertices in each E^i_j of Definition 2(c) until CD_2 or its subgraph is reached.

Step 5: Assign a 0 or 1 to each vertex of CD_2. Then substitute it into the template of each vertex of G. The result is the second factor for the vertex. The concatenation of this factor and that from Step 4 is the code word for the vertex.

Procedure Template Generation

* *This is a procedure to generate templates for (the factors of the code words of) the vertices of* G *in terms of those for the vertices of* G^q*, when* G^q *exists.* *\

Step 1: Assign an arbitrary template to each vertex of G^{q+1}. But no two of them may be identical. Step 2: Obtain a template for each vertex of G^q, $q \geq 0$, in terms of templates for G^{q+1} as follows:

Let E^{q+1} be a partition of the vertex set of G^q which satisfies the conditions of Definition ??. Then each vertex of G^{q+1} corresponds to a block of E^{q+1}. Thus a vertex corresponding to a block E_i^{q+1} is given E_i^{q+1} as its template. For each vertex u of G^q, if it is in block E_i^{q+1} and if it is connected to vertices of E_j^{q+1}, then the template for u in terms of vertices of G^{q+1} is $E_i^{q+1}E_j^{q+1}$.

Step 3: The templates for the vertices of G in terms of vertices of G^{q+1} are obtained as follows:

Let $E_{i_1}^q\ E_{i_2}^q\ ...\ E_{i_q}^q$ be the template of a vertex u of G in terms of templates for G^q. Also let $E_{i_j1}^{q+1}E_{i_j2}^{q+1}$ be the representation of $E_{i_j}^q$ in terms of templates for G^{q+1}.
Then simultaneously replace each $E_{i_j}^q E_{i_{j+1}}^q$ with $E_{i_j1}^{q+1}E_{i_j2}^{q+1}E_{i_{j+1}2}^{q+1}$
to obtain the template for u in terms of those for G^{q+1}.

As an example to illustrate Algorithm SRR let us consider the graph G given by the following table.

Source	Destinations	Source	Destinations
0	$0, 1, 4, 5$	16	$0, 1, 4, 5$
1	$2, 3, 6, 7$	17	$2, 3, 6, 7$
2	$0, 1, 4, 5$	18	$0, 1, 4, 5$
3	$2, 3, 6, 7$	19	$2, 3, 6, 7$
4	$8, 9, 12, 13$	20	$8, 9, 12, 13$
5	$10, 11, 14, 15$	21	$10, 11, 14, 15$
6	$8, 9, 12, 13$	22	$8, 9, 12, 13$
7	$10, 11, 14, 15$	23	$10, 11, 14, 15$
8	$16, 17, 20, 21$	24	$16, 17, 20, 21$
9	$18, 19, 22, 23$	25	$18, 19, 22, 23$
10	$16, 17, 20, 21$	26	$16, 17, 20, 21$
11	$18, 19, 22, 23$	27	$18, 19, 22, 23$
12	$24, 25, 28, 29$	28	$24, 25, 28, 29$
13	$26, 27, 30, 31$	29	$26, 27, 30, 31$
14	$24, 25, 28, 29$	30	$24, 25, 28, 29$
15	$26, 27, 30, 31$	31	$26, 27, 30, 31$

Then V can be partitioned into the following subsets which satisfy the conditions of Definition ??:
$A_0 = \{0, 1, 4, 5\}, A_1 = \{2, 3, 6, 7\}, A_2 = \{8, 9, 12, 13\}, A_3 = \{10, 11, 14, 15\}, A_4 = \{16, 17, 20, 21\}, A_5 = \{18, 19, 22, 23\}, A_6 = \{24, 25, 28, 29\}, A_7 = \{26, 27, 30, 31\}$.
G^1 in this case is given by the following table.

Source	Destinations	Source	Destinations
A_0	A_0, A_1, A_2, A_3	A_4	A_0, A_1, A_2, A_3
A_1	A_0, A_1, A_2, A_3	A_5	A_0, A_1, A_2, A_3
A_2	A_4, A_5, A_6, A_7	A_6	A_4, A_5, A_6, A_7
A_3	A_4, A_5, A_6, A_7	A_7	A_4, A_5, A_6, A_7

Then V^1 can be partitioned into the following subsets which satisfy the conditions of Step 1(b) of Algorithm SRR:
$B_0 = \{A_0, A_1, A_2, A_3\}$, $B_1 = \{A_4, A_5, A_6, A_7\}$.
G^2 in this case is given by the following table.

Source	Destinations
B_0	B_0, B_0, B_1, B_1
B_1	B_0, B_0, B_1, B_1

Here all parallel arcs are listed. In G^2 these parallel arcs are coalesced into one arc. So G^2 is, in this case, CD_2.

As can be easily seen from above, the code word of a vertex of G has two factors. It can be obtained as follows.

First the templates for the vertices of G are given in terms of vertices of G^1 as given in the following table.

Vertex	Code	Vertex	Code	Vertex	Code	Vertex	Code
0	A_0A_0	8	A_2A_4	16	A_4A_0	24	A_6A_4
1	A_0A_1	9	A_2A_5	17	A_4A_1	25	A_6A_5
2	A_1A_0	10	A_3A_4	18	A_5A_0	26	A_7A_4
3	A_1A_1	11	A_3A_5	19	A_5A_1	27	A_7A_5
4	A_0A_2	12	A_2A_6	20	A_4A_2	28	A_6A_6
5	A_0A_3	13	A_2A_7	21	A_4A_3	29	A_6A_7
6	A_1A_2	14	A_3A_6	22	A_5A_2	30	A_7A_6
7	A_1A_3	15	A_3A_7	23	A_5A_3	31	A_7A_7

Next the templates for the vertices of G^1 are represented by vertices of G^2 as given in the following table. Note that some vertices have identical template. Their code words must therefore be distinguished by introducing another factor.

Vertex	Code	Vertex	Code
A_0	B_0B_0	A_4	B_1B_0
A_1	B_0B_0	A_5	B_1B_0
A_2	B_0B_1	A_6	B_1B_1
A_3	B_0B_1	A_7	B_1B_1

To find the code word for A_is, assign 0 to B_0 and 1 to B_1, and distinguish A_is with the same template by assigning a second factor of 0 or 1. Then we have for A_is

Vertex	Code	Vertex	Code
A_0	(00)(0)	A_4	(10)(0)
A_1	(00)(1)	A_5	(10)(1)
A_2	(01)(0)	A_6	(11)(0)
A_3	(01)(1)	A_7	(11)(1)

Applying to these the templates for the vertices of G, the following code words are obtained for the vertices of G¿

Vertex	Code	Vertex	Code	Vertex	Code	Vertex	Code
0	(000)(00)	8	(010)(00)	16	(100)(00)	24	(110)(00)
1	(000)(01)	9	(010)(01)	17	(100)(01)	25	(110)(01)
2	(000)(10)	10	(010)(10)	18	(100)(10)	26	(110)(10)
3	(000)(11)	11	(010)(11)	19	(100)(11)	27	(110)(11)
4	(001)(00)	12	(011)(00)	20	(101)(00)	28	(111)(00)
5	(001)(01)	13	(011)(01)	21	(101)(01)	29	(111)(01)
6	(001)(10)	14	(011)(10)	22	(101)(10)	30	(111)(10)
7	(001)(11)	15	(011)(11)	23	(101)(11)	31	(111)(11)

Let us now prove that a sequential machine is realizable with two shift registers if and only if Algorithm SRR produces a code for its state diagram.

Theorem 10 *A sequential circuit with a state transition graph G is realizable with two shift registers if Algorithm SRR produces a coding for its states.*

This theorem is going to be proved by a series of lemmas.

Lemma 2 *The vertices of G^q are coded by pairs of vertices of G^{q+1} such that no two are coded with the same pair.*

Proof: If G^{q+1} exists, then there is a partition E^{q+1} of the vertex set of G^q which satisfies the conditions of Definition ??. Hence a vertex of G^q is in exactly one of the blocks of E^{q+1}. Furthermore by(2)(b) of Definition ?? the vertices in each block are connected to vertices of different blocks of E^{q+1} except for case 1(b) of Algorithm SRR. Hence for each vertex u, if it is in block E_i^{q+1} and if it is connected to vertices of E_j^{q+1}, then it is given a code word $E_i^{q+1}E_j^{q+1}$. Furthermore, in case 1(b), each pair of vertices having an identical template are given additional string(s) of 0 or 1 to distinguish them. Hence this coding clearly distinguishes the vertices of G^q. ♣

Lemma 3 *Let $E_{i_1}^q E_{i_2}^q ... E_{i_q}^q$ be the template of a vertex u of G in terms of vertices of G^{q-1}. Also let $E_{i_j1}^{q+1}E_{i_j2}^{q+1}$ be the representation of $E_{i_j}^q$ in terms of vertices of G^{q+1}.*
Then $E_{i_j2}^{q+1} = E_{i_{j+1}1}^{q+1}$

Proof: If $E_{i_j}^q = E_{i_j1}^{q+1}E_{i_j2}^{q+1}$ and $E_{i_{j+1}}^q = E_{i_{j+1}1}^{q+1}E_{i_{j+1}2}^{q+1}$, then $E_{i_j}^q$ belongs to $E_{i_j1}^{q+1}$ and $E_{i_{j+1}}^q$ belongs to $E_{i_{j+1}1}^{q+1}$.

Since there is an arc from $E_{i_j}^q$ to $E_{i_{j+1}}^q$ in G^{q-1}, this means that $E_{i_j}^q$ is connected to $E_{i_{j+1}1}^{q+1}$ by an arc of G^q. Hence $E_{i_j2}^{q+1} = E_{i_{j+1}1}^{q+1}$. ♣

This lemma implies that if the vertices of G are coded by q-tuples of vetices of G^{q-1}, then they can be coded by $(q+1)$-tuples of vertices of G^q.

Lemma 4 *The template of the destination of a transition is obtained by shifting that of the source to the left one position.*

Proof: It is going to be shown that if a transition from u to v exists in G, and if the template of v in terms of vertices of G^q is obtained by shifting that of u to the left one position, then the template of v in terms of vertices of G^{q+1} is obtained by shifting that of u to the left one position.

Let $E_{i_1}^{q+1}E_{i_2}^{q+1}...E_{i_{q+1}}^{q+1}$ be the template of a vertex u of G in terms of vertices of G^q. Also let $E_{j_1}^{q+1}E_{j_2}^{q+1}...E_{j_{q+1}}^{q+1}$ be the template of a vertex v of G in terms of vertices of G^q. Then for every p, $2 \le p \le q$, $E_{i_p}^{q+1}E_{i_{p+1}}^{q+1} = E_{j_p}^{q+1}E_{j_{p+1}}^{q+1}$, because $E_{i_l}^{q} = E_{i_l}^{q+1}E_{i_{l+1}}^{q+1}$, $E_{j_l}^{q} = E_{j_l}^{q+1}E_{j_{l+1}}^{q+1}$, for $1 \le l \le q$, and $E_{j_1}^{q}E_{j_2}^{q}...E_{j_q}^{q}$ is obtained by shifting $E_{i_1}^{q}E_{i_2}^{q}...E_{i_q}^{q}$ to the left one position. Hence by shifting $E_{i_1}^{q+1}E_{i_2}^{q+1}...E_{i_q}^{q+1}$ to the left one position and shifting in $E_{j_q}^{q+1}$ from the right, the template for v is obtained. ♣

Proof of Theorem ??: From the algorithm it is obvious that every vertex of G is given a code word consisting of two factors (strings of 0 and 1). By Lemma **??** it can easily be seen that no two vertices of G have identical code word (proof by contradiction). Finally by Lemma **??**, all the transitions are represented by a shift of a code word. ♣

The converse of Theorem **??** also holds as shown below.

Theorem 11 *If G is realizable with two shift registers of length k_1+1 and k_2+1, $k_1 \ge k_2$, then*
(1) G^i exists for all i, $1 \le i \le k_2$,
(2) if $k_1 = k_2$, then G^{k_2} is CD_4,
(3) if $k_1 > k_2$, then
(a) for every vertex of G^{k_2} all of its outgoing arcs go to at most two different vertices(destinations),
(b) $(G^{k_2})^i$ exists for $1 \le i \le (k_1 - k_1)$, where each block contains at most two vertices of $(G^{k_2})^{i-1}$, and
(c) $(G^{k_2})^{k_1-k_2}$ is CD_2.
Hence if G is realizable with two shift registers, then Algorithm SRR applied on G produces CD_2 or CD_4.

Proof: Suppose that the vertices of G are coded with binary (k_1+1)-tuples and (k_2+1)-tuples as factors, where $k_1 \ge k_2$.

Given G, construct G^i for i, $1 \leq i \leq k_2$, as follows: First, construct a partition E^i of vertices of G^{i-1} so that vertices u and v of G are in the same block of the partition E^i if and only if their code words differ only in the rightmost bit of each of its factors.

Then let V^i be the set of the blocks of partition E^i of the vertex set V^{i-1} of G^{i-1}. For a vertex $u \in V^i$, construct its code word by removing the rightmost bit of each factor of the code word for a vertex of V^{i-1} in u.

(1) Since each transition corresponds to a shift left of each factor of a code word, (2)(a) of Definition ?? is satisfied.
Since the code words of the vertices in each block differ only in the rightmost bit of each factor, there can be at most four of them. Hence (c) of Definition ?? holds.
With this construction of G^i, if $k_1 \neq k_2$, then one of the factors disappears for G^{k_2} on. Hence for G^i, $i \geq k_2$, at most two vertices can be in each block.

The code words of vertices of G^{i-1} in a vertex u of G^i differ in the rightmost bit of at least one of its two factors. Hence their destinations are in different vertices of G^i if $i \leq k_2$. Hence Definition ?? (b) is satisfied. Hence (1) holds.

(2) According to the way G^i is constructed, the vertices of G^{k_2} have a code word consisting of two 1-bit factors. Hence there are at most four vertices in G^{k_2}. Also since any 1-bit factoe can shift to any other 1-bit factor, each vertex of G^{k_2} can be connected to any other. Hence G^{k_2} is CD_4 or its subgraph.

Use the method of constructing G^i and code words used for $i \leq k_2$ to construct G^i for $i > k_2$.

(3) Construct G^i the same way as for (1) except that the code words here consist of one factor instead of two.
(3)(a) If $i = k_2 + 1$, since the code word for the vertices of G^i consists of one factor, the vertices of V^{i-1} in a vertex $u \in V^i$ can go to at most two different blocks.

(3)(b) Since the code words of the vertices in each block differ only in the rightmost bit, there can be at most two of them.

(3)(c) According to the way G^i is constructed, the vertices of $(G^{k_2})^{k_1-k_2}$ are coded by a 0 or 1. Hence there are two vertices in $(G^{k_2})^{k_1-k_2}$. Since both 0 and 1 can be reached from either of 0 and 1 by shift, $(G^{k_2})^{k_1-k_2}$ is CD_2. ♣

7 General Case

Definition 6 *A digraph G having the following properties is denoted by $DM(k_1, ..., k_m)$, where k_i are nonnegative integers and $k_i \geq k_{k+1}$ for $1 \leq i \leq m-1$:*
1. G has $2^{(k_1+...+k_m)}$ vertices.
2. The vertices can be labeled with a concatenation of binary k_1-tuple, ... , k_m-tuple as factors such that the labels are distinct from each other.
3. For every pair of vertices u and v of G, there is an arc (u, v) in G, if and only if the label of v is obtaind by shifting each factor of the label of u to the left by one bit and shifting in 0 or 1 from the right.

Note that according to this definition $DM(k_1, ..., k_p, 0, ..., 0) = DM(k_1, ..., k_p)$.

Then $DM(k_1, ..., k_m)$ has the properties given by the following theorems.

Definition 7 *A* k-th order homomorph G^k *of a graph G is defined recursively as follows:*

(1) $G^0 = G$.

(2) For i = 1 to k, partition the vertex set V^{i-1} of G^{i-1}(denote the partition by E^i) so that

(a) for any two vertices u and $v \in V^{i-1}$, if $(u, v) \in T_r$, then u and v are in one block of the partition E^i,

(b) for each block E^i_j of partition E^i if u and $v \in E^i_j$, then the vertices to which u and v are connected in G^{i-1} are in different blocks of E^i, and

(c) each block E^i_j contains at most 2^m vertices of G^{i-1} for some positive integer m.

If no such partition exists, then G^i does not exist.

(3) Construct G^i by using a vertex for each block of the partition E^i and by connecting vertices x and y of G^i by an arc from x to y iff (u, v) is an arc of G^{i-1}, where u and v are vertices in the blocks of E^{i-1} represented by x and y, respectively.

Theorem 12 *Let G be a $DM(k_1, ..., k_m)$. Also for any vertex u of G, let u also denote its label. Then for any arc (u, v) of G,*

$$v = 2u + c_1 + \Sigma_{i=2}^{m} c_i 2^{\Sigma_{j=i}^{m} k_j},$$

where $c_i = 0$ or 1 for all i, $1 \leq i \leq m$, and the labels are considered as a binary number of $\Sigma_{i=1}^{m} k_i$ bits.

Proof: If each factor of the label of u is shifted left one bit and 0 is shifted into the rightmost bit, then the resultant label is $2u$. Further if 1 is shifted into the k_i-th factor, then $2^{\Sigma_{j=i}^{m} k_j}$ is added to $2u$. Since 1 can be shifted into any number of factors of the code word of u, the formula for v holds. ♣

Theorem 13 *$DM(k_1-1, ..., k_m-1)$ is a first homomorph of $DM(k_1, ..., k_m)$.*

Proof: Partition the vertex set of $DM(k_1, ..., k_m)$ such that vertices u and v are in the same block if and only if their labels differ only in the rightmost bit of some of the factors of their respective code words.
Then construct a graph H by using a vertex for each block of the partition and by connecting vertices x and y of H by an arc from x to y if and only if (u, v) is an arc of $DM(k_1, ..., k_m)$, where u and v are vertices in the blocks represented by x and y, respectively.
Then it can be easily seen that H is a first homomorph of $DM(k_1, ..., k_m)$, and it is $DM(k_1-1, ..., k_m-1)$. ♣

Remark 1 *$DM(k_1, ..., k_m) = CD_{2^m}$, if $k_1 = ... = k_m = 1$.*

Proof: Firstly since the vertices of $DM(k_1, ..., k_m)$ are labeled with a string of m 0's and 1's, there are 2^m vertices. Secondly since each code word can be shifted to 2^m code words, each vertex of $DM(k_1, ..., k_m)$ is connected to 2^m vertices including itself. Hence the graph $DM(k_1, ..., k_m)$ is CD_{2^m}. ♣

Theorem 14 *If $k_i = 1$ for $i \geq p+1$, then the vertex set of $DM(k_1, ..., k_p, ..., k_m)$ can be partitioned as follows:*
(1) Each block has 2^{m-p} vertices, and
(2) if each block of vertices are coalesced into one vertex also coalescing resulting parallel arcs, the resulting graph is $DM(k_1, ..., k_p)$.

Proof: Construct a partition E of the vertex set of $DM(k_1, ..., k_p, ..., k_m)$ such that two vertices u and v are in the same block of E if and only if their code words differ in the last $m-p$ factors.

It can easily be seen that E satisfies (1) and (2) ♣

Algorithm GSRR

Input: A state transition graph G of a sequential circuit
Output: A code word given as a concatenation of factors for each vertex of G such that the code word of the destination state of each transition is obtained by shifting the factors of the code word of the source state to the left one position.

Step 1: Find the i-th order homomorph G^i, $i \geq 1$, of G successively and apply **Procedure Template Generation** for each G^i starting with G until one of

the following two graphs is reached:
(a) CD_{2^m} or its subgraph for some m
(b) For every vertex of G^i, all of its outgoing arcs go to at most 2^p different vertices(destinations). (Hence if the out-degree is greater than 2, then there are as many as 2^{m-p} parallel arcs between a pair of vertices in G^i and the vertices of G^{i-1} represented by parallel arcs get the same template. Here $DM(k_1 - k_m, ..., k_p - k_m, 1, ..., 1)$ or its subgraph is reached). Let us denote this homomroph by G^K.

If (a) is the case, then go to Step 2, else if (b), then go to Step 3. If neither (a) nor (b), then G can not be realized with two shift registers.

Step 2: Assign arbitrarily a string of length m of 0's and 1's to the 2^m vertices of CD_{2^m}. No two vertices receive the same string. These strings are interpreted as m 1-bit factors. Obtain m factors for each vertex of G by substituting each of these 1-bit factors into the templates for CD_{2^m}. The code word for a vertex of G is the concatenation of these m factors.

Step 3: At this point i.e. at the end of 1(b), $m - p$, $p < m$, factors of the code word for each vertex of G can be found as follows:
To each of the 2^{m-p} vertices of G^K having an identical template, i.e. the destinations of parallel arcs, give a string of length $m - p$ consisting of 0's and 1's to distinguish them from each other. Assign to all the other vertices a string of length $m-p$ consisting of 0's and 1's arbitrarily. This string is considered $m-p$ factors of length 1. $m - p$ factors of the code word of a vertex of G can now be found as follows: For each bit of the string for each vertex of G, substitute it into the template. Concatenate $m - p$ factors thus obtained for each vertex. The result is a portion of the code word for the vertex. Concatenate it with the template, that is the one which is identical for all of 2^{m-p} vertices.

To find the remaining portion of the code word for the vertices of G, go to 4.

Step 4: Merge parallel arcs, hence their destination vertices.
Then continue finding i-th order homomorph of G^K and templates but with at most 2^p vertices in each E^i_j of Definition 2(c) until $DM(k'_1, ..., k'_q, 1, ..., 1)$ or its subgraph is reached.

Step 5: Repeat Steps 3 and 4 until CD_{2^t} or its subgraph for some t is reached.

Step 6: Assign a string of length t consisting of 0's and 1's to each vertex of CD_{2^t}. Then apply the template to each bit of this string to find the last t factors for each vertex of G. The concatenation of these factors and those from Step 5 is the code word for a vertex of G.

Procedure Template Generation

{This is a procedure to generate templates for (the factors of the code words of) the vertices of G in terms of those for the vertices of G^q, when G^q exists.}

Step 1: Assign an arbitrary template to each vertex of G^{q+1}. But no two of them may be identical. Step 2: Obtain a template for each vertex of G^q, $q \geq 0$, in terms of templates for G^{q+1} as follows:

Let E^{q+1} be a partition of the vertex set of G^q which satisfies the conditions of Definition ??. Then each vertex of G^{q+1} corresponds to a block of E^{q+1}. Thus a vertex corresponding to a block E_i^{q+1} is given E_i^{q+1} as its template. For each vertex u of G^q, if it is in block E_i^{q+1} and if it is connected to vertices of E_j^{q+1}, then the template for u in terms of vertices of G^{q+1} is $E_i^{q+1} E_j^{q+1}$.

Step 3: The templates for the vertices of G in terms of vertices of G^{q+1} are obtained as follows:

Let $E_{i_1}^q\ E_{i_2}^q\ ...\ E_{i_q}^q$ be the template of a vertex u of G in terms of templates for G^q. Also let $E_{i_j 1}^{q+1} E_{i_j 2}^{q+1}$ be the representation of $E_{i_j}^q$ in terms of templates for G^{q+1}.
Then simultaneously replace each $E_{i_j}^q E_{i_{j+1}}^q$ with $E_{i_j 1}^{q+1} E_{i_j 2}^{q+1} E_{i_{j+1} 2}^{q+1}$
to obtain the template for u in terms of those for G^{q+1}.

Theorem 15 *A sequential circuit with a state transition graph G is realizable with m shift registers, if Algorithm GSRR produces a coding for the vertices of G.*

Proof: The same lemmas and their proof as for the two shift registers case are also valid for this case. Hence this theorem also holds. ♣

Theorem 16 *Suppose that G is realizable with m shift registers of various lengths. Suppose also that r_j of the shift registers have the same length $k_{i_j}+1$, for $1 \leq j \leq l$, and $\Sigma_{j=1}^{l} r_j = m$. Assume without loss of generality that $k_{i_j} \geq k_{i_{j+1}}$ for all j.*
Then Algorithm GSRR, when applied rto G, produces $CD_{2^{r_j}}$, $j = l, ..., 1$, in that order.

Proof: Since $G = DM(k_1, ..., k_m)$, by Theorem ??, G^i exists for i, $i < k_m$, and $G^i = DM(k_1 - i, ..., k_m - i)$.
Hence $DM(k_1 - k_m, ..., 1, ..., 1)$ is eventually reached. If it is $DM(1, ..., 1)$, then DM_{2^m} has been obtained. If it is not $DM(1, ..., 1)$, then Step 1(b) of Algorithm

GSRR applies, and by Theorem ??, $DM(k_1 - k_m, ..., k_{m-r_l} - k_m)$ is obtained.

Algorithm GSRR is now applied recursively and $CD_{2^{r_j}}$ are produced for the rest of j's, i.e. $j = l - 1, ..., 1$.
♣

Thus if Algorithm GSRR does not produce a coding for G, then G is not realizable with shift registers.

8 Conclusion

References

[1] V. D. Agrawal and K.-T. Cheng. Finite state machine synthesis with embedded test function. *Journal of Electronic Testing:Theory and Applications*, pages 221–228, 1990.

[2] T. F. Arnold, C.-J. Tan, and M. M. Newborn. Iteratively realized sequential circuits. *IEEE Trans. Comput.*, pages 54–66, January 1970.

[3] C. Chao and Jr. H. H. Loomis. High rate realization of finite-state machines. *IEEE Trans. Comput.*, pages 729–740, July 1975.

[4] K. T. Cheng and V. D. Agrawal. Design of sequential machines for efficient test generation. *ICCAD*, pages 358–361, 1989.

[5] A. D. Friedman. Feedback in synchronous sequential switching circuits. *IEEE Trans. Elect. Comput.*, pages 354–367, June 1966.

[6] H. Fujiwara and K. Kinoshita. Design of diagnosable sequential machines utilizing extra outputs. *IEEE Trans. Comput.*, pages 138–145, February 1974.

[7] H. Fujiwara, Y. Nagao, T. Sasao, and K. Kinoshita. Easily testable sequential machines with extra inputs. *IEEE Trans. Comput.*, pages 821–826, August 1975.

[8] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. Comput.*, pages 1137–1144, December 1983.

[9] F. G. Gray, L. C. C. Shih, and R. A. Thompson. Diagnosis of faults in modular trees. *IEEE Trans. Comput.*, pages 342–353, May 1979.

[10] D. R. Haring. On shift-register realizations of sequential machines and finite-state universal sequential machines. *IEEE Trans. Comput.*, pages 309–312, April 1968.

[11] J. Hartmanis. Loop-free structure of sequential machines. *Information and Control*, 5:25–43, 1962.

[12] J. Hartmanis. On the state assignment problem for sequential machines. I. *IRE Trans. Elect. Comput.*, pages 157–165, June 1961.

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

[14] J. Hartmanis and R. E. Stearns. On the state assignment problem for sequential machines. II. *IRE Trans. Elect. Comput.*, pages 593–603, December 1961.

[15] J. Hartmanis and R. E. Stearns. Pair algebra and its application to automata theory. *Information and Control*, pages 485–507, December 1964.

[16] T. Inose, Y. Yamada, E. Tomita, and H. Takahashi. Experimental evaluation of a randomized algorithm for finding a near-maximum clique. In *Proc. IECIE 1993 Fall Conference*, pages (6–3), September 1993(in Japanese).

[17] D. L. Johnson and K. H. O'Keefe. The application of shift registers to secondary state assignment:Part I. *IEEE Trans. Comput.*, pages 954–965, October 1968.

[18] D. S. Johnson, C. H. Papadimitriou, P. Seymour, and M. Yanakakis. The complexity of multi way cuts. *Extended Abstract*, 1983.

[19] S. Kanjilal, S. T. Chakradhar, and V. D. Agrawal. Test function embedding algorithms with application to interconnected finite state machines. *IEEE Trans. on Comput.-Aided Design of Integrated Circuits*, pages 1115–1127, September1995.

[20] C. L. Liu. K-th order finite automaton. *IEEE Trans. Comput.*, pages 470–475, October 1963.

[21] R. L. Martin. *Studies in feedback-shift-register synthesis of sequential machines*. MIT Press, Cambridge, MA, 1969.

[22] A. J. Nichols. Minimal shift-register realizations of sequential machines. *IEEE Trans. Elect. Comput.*, pages 688–700, October 1965.

[23] B. A. Prasad and F. G. Gray. Fault diagnosis in uniform modular realization of sequential machines. In *Proc. of Fault Tolerant Computing Symp.*, pages 157–162, 1973.

[24] W. D. Roome and H. C. Torng. Algorithms for multiple shift register realizations of sequential machines. *IEEE Trans. Comput.*, pages 933–943, October 1973.

[25] K. K. Saluja. Synchronous sequential machines: a modular and testable design. *IEEE Trans. Comput.*, pages 1020–1025, November 1980.

[26] M. Schulz and E. Auth. Advanced automatic test pattern generation and redundancy identification techniques. In *Proc. of Fault Tolerant Computing Symp.*, pages 30–35, 1988.

[27] C. C. Su and S. S. Yau. Unitary shift-register realizations of sequential machines. *IEEE Trans. Comput.*, pages 312–324, April 1968.