September 1996

UILU-ENG-96-2220
CRHC-96-12

University of Illinois at Urbana-Champaign

Reliability Driven Synthesis of Sequential Circuits

Frank F. Hsu, Michael S. Hsiao, and Prithviraj Banerjee

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801

SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE	*Form Approved OMB NO. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1996	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Reliability Driven Synthesis of Sequential Circuits	5. FUNDING NUMBERS
6. AUTHOR(S) Frank F. Hsu, Michael S. Hsiao, and Prithviraj Banerjee	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Coordinated Science Laboratory University of Illinois 1308 W. Main St. Urbana, IL 61801	8. PERFORMING ORGANIZATION REPORT NUMBER (CRHC-96-12)
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Semiconductor Research Corp. P.O. Box 12053 Research Triangle Park, NC 27709-2053	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

While conventional logic synthesis tools try to minimize the area of the circuit implementation, recent research has been focused on logic synthesis with other objectives, such as maximizing the performance and testability of the circuits. In this paper, we propose sequential logic synthesis with a different objective, namely to increase the reliability of the circuit; we propose procedures to synthesize fault-secure sequential circuits. With a given state transition graph description of a sequential machine, this procedure will produce a fault-secure logic implementation of the machine. This logic level implementation is capable of detecting any single fault in the circuit, assuming the primary inputs are fault-free.

Two synthesis schemes for generating sequential circuits with the ability to detect single stuck-at faults are studied in this paper. The first scheme uses simple duplication of an FSM augmented with a totally self-checking (TSC) comparator for fault detection.
For the second scheme, we utilize a modified state assignment, parity code, and Berger code to detect any single fault in the circuit. The state assignment uses a minimum distance two code for single fault detection in the next state logic (NSL). All of the TSC checkers used in the circuits are capable of detecting faults within themselves, resulting in a sequential machine capable of detecting any single fault within it.

14. SUBJECT TERMS reliability, logic synthesis, self-checking, fault-secure	15. NUMBER IF PAGES 17
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

RELIABILITY DRIVEN SYNTHESIS OF SEQUENTIAL CIRCUITS

Frank F. Hsu, Michael S. Hsiao, and Prithviraj Banerjee

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
1308 W. Main Street
University of Illinois
Urbana, IL 61801

Abstract

While conventional logic synthesis tools try to minimize the area of the circuit implementation, recent research has been focused on logic synthesis with other objectives, such as maximizing the performance and testability of the circuits. In this paper, we propose sequential logic synthesis with a *different objective*, namely to increase the reliability of the circuit. We propose procedures to synthesize fault-secure sequential circuits, that is, circuits which under faults will either produce the correct outputs, or will flag an error if the output is incorrect. With a given state transition graph description of a sequential machine, this procedure will produce a fault-secure logic implementation of the machine. This logic level implementation is capable of detecting any single fault in the circuit, assuming the primary inputs are fault-free.

Two synthesis schemes for generating sequential circuits with the ability to detect single stuck-at faults are studied in this paper. The first scheme uses simple duplication of an FSM augmented with a totally self-checking (TSC) comparator for fault detection. The generation of the duplicated circuit and the appropriate comparators are integrated into the synthesis system.

For the second scheme, we utilize a modified state assignment, parity code, and Berger code to detect any single fault in the circuit. The state assignment uses a minimum distance two code for single fault detection in the next state logic (NSL). Each next-state variable is generated with an independent logic cone and the output logic is composed of a multilevel logic synthesized under the constraint that every factor shared among various output nodes appears in the positive phase only. All of the TSC checkers used in the circuits are capable of detecting faults within themselves, resulting in a sequential machine capable of detecting any single fault within it.

While the theory for such techniques have been known to the fault-tolerant computing commmunity, the *contribution* of this paper is to integrate those techniques in the framework of sequential logic synthesis, implement such a logic synthesis system as part of the UC Berkeley OCT/VEM tools, and to evaluate the overheads of the schemes after performing a complete place and route using standard cell design.

This research was supported in part by the Semiconductor Research Corporation under contract SRC 95-DP-109 and in part by Joint Services Electronics Program under grant N00014-90-J-1270

1 Introduction

Larger and more complex systems are being put into single chips as the VLSI technology improves. Due to the increase in complexity and decrease in feature size, the circuits tend to be more error-prone, thus reliability of the circuits will become an important issue in VLSI design in the future.

As the VLSI design methodology moves towards higher and higher levels of automation, circuits these days are being increasingly designed with the aid of synthesis tools at the register transfer level, logic level and physical design level. In the future, VLSI designers will only specify high level objectives, namely area minimization, delay optimization, testability and reliability improvement, and the synthesis tools will automatically generate the appropriate circuits, and produce the masks for fabricating the VLSI chip.

While conventional logic synthesis tools try to minimize the area of the circuit implementation[3, 5], recent research has been focused on logic synthesis with other objectives, such as maximizing the performance[6] and testability of the circuits[7]. In this paper, we propose a sequential logic synthesis with a *different objective* of increasing the reliability of the circuit.

Numerous techniques are known to the fault tolerant computing community for increasing the reliability of VLSI circuits. All of them use some form of redundancy such as information redundancy, hardware redundancy, and time redundancy [1]. These techniques enable the circuits to detect and/or tolerate faults in them. Information redundancy involves the coding of information such that error detecting and correcting codes are incorporated into the data bits. Codes such as parity check codes, cyclic codes, checksum codes, Berger codes, and numerous others have been proposed for reliable data transfer and storage [2].

Hardware redundancy is an alternative way to increase the reliability of circuits. The simplest form of hardware redundancy involves duplication of the functions, and comparing the results of the two copies by an equality checker at the outputs. Some advantages of duplication are that it is applicable to most functions in general, and that duplication is simple to implement. An obvious disadvantage of this simple technique is the area overhead. More than 100% overhead is always needed for duplication due to the cost of the comparator, which may not be feasible to incorporate into regular VLSI chips.

The third option, time redundancy, is used when the performance of the circuit is not a crucial issue. Time redundancy involves applying the same inputs to the same hardware repeatedly in time to compare the results. Recomputing with shifted operands and alternating logic are examples of this approach [1].

In summary, a large set of techniques are available in the fault tolerant computing area which can be applied to make a VLSI system reliable. Unfortunately, in their most general form (duplication and comparison), the area overheads are too high (more than 100%), though lower cost custom tailored techniques are available for specific functions [2].

Ensuring the reliability of combinational circuits involves only the checking of primary outputs. To ensure the reliability of sequential circuits, on the other hand, is more complex due to the dependence of internal states. When an error occurs, it is crucial to make sure that the next state is correct in addition to the outputs. In other words, if a fault is present, it is desirable to detect the fault as soon as possible, preferably in the current state instead of some states later.

In this paper, we propose procedures to synthesize fault-secure sequential circuits: circuits that under faults will either produce the correct outputs, or will flag an error if the output is incorrect. With a given state transition graph description of a sequential machine, this procedure will produce a fault-secure logic implementation of the circuit. This logic level implementation is capable of detecting any single fault in the circuit, assuming the primary inputs are fault-free.

Two synthesis schemes for generating sequential circuits with the ability to detect single stuck-at faults are studied in this paper. The first scheme uses simple duplication of a given sequential circuit augmented with a totally self-checking (TSC) comparator for fault detection. The generation of the duplicated circuit and the appropriate comparators are integrated into the synthesis system.

For the second scheme, we utilize a modified state assignment, parity code, and Berger code to detect any single fault in the circuit. The state assignment utilizes a minimum distance two code for single fault detection in the next state logic. Each next-state variable is generated with an independent logic cone so that a fault in the logic will not affect more than one bit in the next state. Errors in the present state are checked by a TSC parity checker, thus any single fault in the state variables can be detected. The output logic consists of a multilevel logic which is synthesized under the contraint that every factor that is shared among various output nodes appears in the positive phase only. All of the TSC checkers used in the circuits are capable of detecting faults within themselves. The resulting sequential machine will signal errors if any single fault occurred in any part of the sequential machine.

While the theory for such techniques have been known to the fault-tolerant computing commmunity, the *contribution* of this paper is to integrate those techniques in the framework of sequential logic synthesis, implement such a logic synthesis system as part of the UC Berkeley OCT/VEM tools, and to evaluate the overheads of the schemes after performing a complete place and route using standard cell design. The overhead of the two schemes are studied by applying the synthesis techniques to several benchmark sequential circuits.

The paper is organized as follows. Section 2 summarizes previous work in the area. Section 3 includes some of the preliminaries for this paper. Section 4 provides an overview of the synthesis methods using duplication. Some of the implementation issues and methods of the second scheme are described in Section 5. Results of the implementation are described in Section 6.

2 Previous Work

Fault-tolerant sequential machines have been studied as early as in [11]. In that paper, methods were presented for utilizing error correction codes in synchronous systems to improve reliability and gain fault-tolerance. Later in [12], Russo showed an example of a fault tolerant machine by the use of distance-3 codes. Then in [13], Meyer investigated a technique for synthesis of fault tolerant state-assigned machines which resumed the original behavior even if faults occured in the system in some limited manner. In all these methods, the basic approach proposed is to use a distance-3 codes for the state encoding, and having each state variable implemented by an independent logic circuit so that a single fault in the circuit will only produce an error in a single bit position of the state variables. Hence, the single fault can be masked by the decoder which decodes the state variables.

In [14, 15], Sengupta et al discuss single and multiple fault tolerance in sequential machines, and the idea of fail-safe realization for a synchronous sequential machine. Effective linear error-correcting codes are used for the realization of fault-tolerant sequential machines.

Leveugle et al [9, 10] have proposed a design methodology for finite-state machine design using a distance-3 encoding with intrinsic fault tolerant and recovery capabilities, where in the presence of a single bit error produced by a single fault, the state decoder corrects the error with the use of distance-3 codes. It is again assumed that each state variable is produced by an independent logic cone. More recently, this approach has been extended to synthesize such circuits automatically [4].

All of the above work is targeted towards error correction, i.e. fault tolerant finite-state machine design, whereas our work simply deals with fault/error detection.

In [20], Devadas et al have shown an approach for synthesizing irredundant output logic and next-state logic. By having independent input cones implementing the logic circuits of the next state variables, a minimum distance two code will ensure that any single fault in the next-state logic will produce an invalid state at the state variables. This work was targeted to design testable finite-state machines, whereas we are interested in the design of online fault detection methods.

Wang and Jha [21] have proposed different methods for self-checking combinational and sequential circuits. The combinational circuits were generated as inverter-free networks. The primary outputs were encoded with the Berger code, and the state outputs were encoded with an m-out-of-n code. Since the state encoding uses m-out-of-n code, this approach does not need the assumption of independent logic cones for each state variable, but assumes the use of inverter-free logic circuits to guarantee unidirectional errors from single stuck-at faults. Experiments were done using synthesis tools to compute the costs of the overhead only in terms of literal counts. Hence, the area increase due to the requirements of additional filp-flops or checkers were not considered.

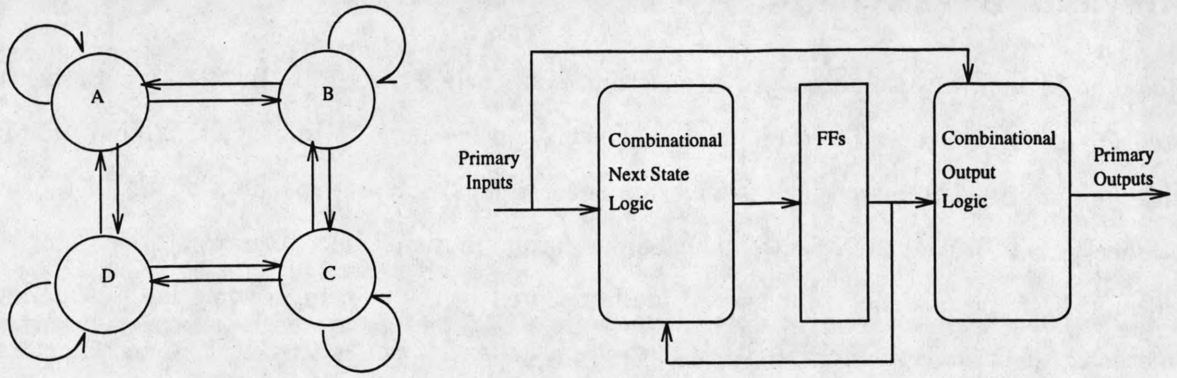

Figure 1: A Mealy machine representation.

De et al [22, 23] have proposed the RSYN synthesis system for generating combinational circuits with error detection capabilities. Two schemes have been proposed, one using a Berger code, the other using a parity code. Experimental results of synthesis using both the schemes on various benchmarks have been presented and compared to a simple scheme using duplication.

The work described in this paper is unique in that (1) it is meant for generating sequential circuits with online fault detection (fault-secure) capability (not for fault tolerant or testable designs) (2) it provides an integrated synthesis system for truly synthesizing reliable circuits as opposed to just the theoretical design (3) experimental evaluation of the actual layout overheads for various benchmark circuits using a standard cell design method by actual placing and routing the cells together with all the checkers, encoders, decoders, etc. (4) it uses a different encoding from previous methods of sequential logic synthesis with fault detection.

3 Preliminaries

A finite state machine (Mealy model) is defined as a quintuple (I,O,S, δ, ω), where I,O,S are respectively the set of inputs, outputs and states, δ: I x S -> S is a state transition function, and ω: I x S -> O is the output function. These machines are described by their state transition graph, and then implemented using a bank of flip-flops to store the current state, and combinational logic for the next state and output function computation as shown in Figure 1.

The classical synthesis of a finite state machine involves three main phases: state assignment, boolean equation generation and logic structure generation[16, 17]. In the first phase, binary codes are assigned to the FSM states. Examples of state encodings are the one-hot encoding (where the state binary codes have as many bits as there are states in FSM, of which only one bit is set to 1), and the compact encoding which uses log(S) bits to denote S states, and performing the state assignment to minimize the estimate of combinational logic required to implement the encoding. Some state assignments are targeted to minimize the combinational logic area assuming two-level implementations, others are targeted to multi-level implementations. The second phase is the generation of boolean equations given the state assignment and the behavior of the FSM Mealy machine model. The third phase is the logic structire generation which includes logic minimization, factorization and technology mapping.

To synthesize a sequential circuit with error detection, it is important to make sure that each part of the circuit can detect the fault when one occurs. Since the next state depends on the primary inputs and the current internal state variables, caution must be taken to ensure that the faults in the circuit will either produce the correct output or yield a invalid result to preserve the fault-secure property [18]. For a sequential circuit to detect any single fault, both the logic to produce the primary output and the logic to generate the next state must have the ability to detect possible errors. We now provide a few definitions related to fault detection in sequential circuits.

A sequential circuit is *self-testing* if, for every fault from a prescribed fault set, the circuit produces a noncode space output for at least one input received during normal operation.

A sequential circuit is *fault-secure* if, for every fault from a prescribed fault set, the circuit never produces an incorrect code space output for normal inputs. More formally, a sequential circuit is fault secure for an input set V $\subseteq$ I, and a fault set F_s if for any input X in I, and for any fault f in F_s, and for any state s in S,, either (1) δ_f (X,q) = δ (X,q) and ω_f (δ_f (X,q)) = ω (δ (X,q)) or (2) ω_f (δ (X,q)) is not in S.

The designs that we will synthesize will be fault-secure sequential circuits.

4 Approach 1: Synthesis using Duplication

The first automated synthesis procedure we have integrated within the logic synthesis system is a complete duplication scheme in which the given circuit of the finite state machine is duplicated by the system automatically. A totally self-checking comparator compares the outputs of the output logic of the two copies of the sequential machines. The synthesis system also automatically generates the circuitry for the TSC comparator which has been parameterized with the number of inputs. The duplication scheme is shown in Figure 2. The error signals (z0 and z1) are dual-rail coded, i.e., valid code words are *01* and *10.* If there is any fault in the comparator or if two signals are not equal, then the error signal will become a non-code word. The TSC comparator using a TSC dual-rail checker is shown in Figure 2, based on the design given in [2].

The dual rail checker checks if a pair of signals is coded in 1-out-of-2 codes, i.e., whether only one of the two signals has the logic value 1. An implementation of 2-bit TSC dual rail checker is also shown in Figure 2 [2]. Two 2-bit signals, a_1a_0 and b_1b_0, are compared for dual-rail property in the circuit shown in the figure. A totally self checking dual rail checker for any width of the signal vector can be implemented by the use of

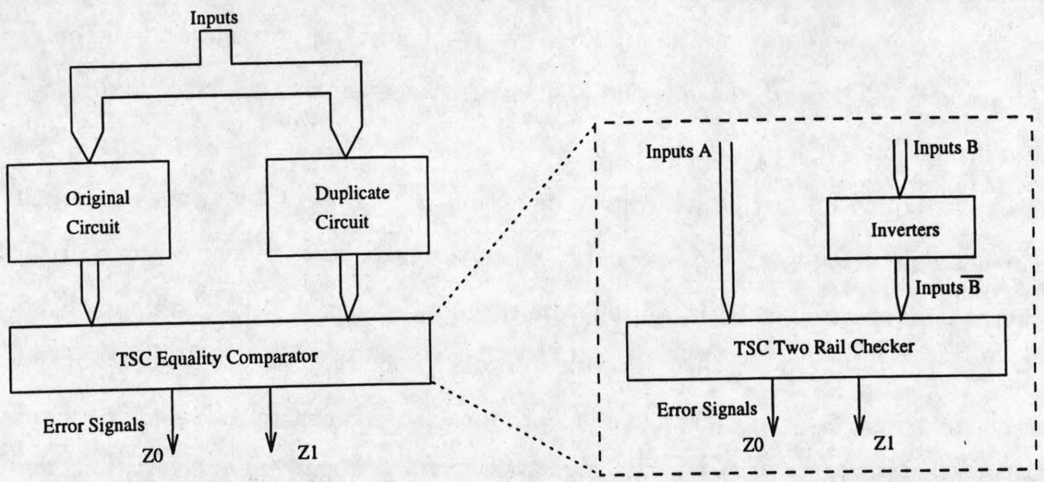

Figure 2: Schematic for the total duplication scheme.

2-bit TSC dual-checker as a building block.

5 Approach 2: Coding Method

In this section, we will propose a second logic synthesis scheme based on the usage of the Berger code on the output vector of a combinational block realizing the output logic of the finite state machine, and the use of a parity code for the state vector.

A block diagram for this proposed fault-secure circuit is shown in Figure 3.

For the logic that generates the next state to detect an error, one solution is to modify the state assignment of the finite state machine so the Hamming distance between any pair of states is at least two [19]. In doing so, any single fault in the state logic will produce an invalid next state. However, one has to make sure that no more than one bit in the flip-flops may be affected simultaneously by a single fault in the input circuitry. Therefore, in order to constrain that a single fault only causes single bit errors in the next state, the input logic to the flip-flops cannot have any sharing of gates. In other words, each flip-flop generated by this design will have its own independent input cone.

We assume a state assignment where the states are encoded with an even number of 1s. Such an encoding guarantees a Hamming distance of two. An even-parity checker can be used to verify the correctness of the flip flops.

We now address the synthesis of the output logic. A Berger code is used to detect errors that may arise in the output logic which can in general have multiple inputs and multiple outputs. The Berger code is a systematic code where given an information vector X, a code vector C(X) is attached to it, where C(X) denotes the count of zeros in the information vector. The Berger code can detect single bit errors and multiple

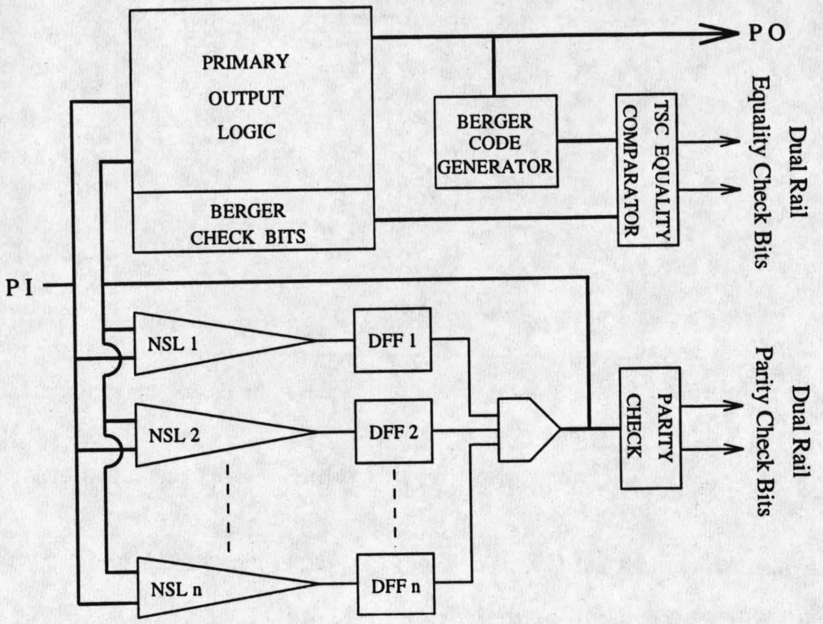

Figure 3: Fault tolerant circuit block diagram

bit unidirectional errors. Due to the sharing of common terms in a multilevel circuit, a single fault in a logic gate of a multilevel circuit can potentially affect multiple outputs. If the fault causes one output to yield an erroneous value 1 instead of the correct value 0 and simultaneously another output to yield an erroneous value 0 instead of the correct value 1, we cannot detect the fault using Berger code. For example, in Figure 4, the fault shown can produce a bi-directional error at the primary outputs O_p and O_q. This is due to the fact that the path to the output O_p does not have any signal inversion, where as the path to the output O_q has one signal inversion. Hence, we have to implement the multilevel circuit in such a way that any single fault in the circuit results in unidirectional errors at the outputs. De et al [22, 23] have proposed an implementation of the multilevel circuits with Berger coded outputs under some constraints. The MIS [3] logic synthesis algorithm is based on algebraic factorization, where given some boolean expressions, common factors are extracted and synthesized in the circuit as a node in the boolean network. Then the expressions which need that common factor use the new node instead of implementing that common factor many times. In the MIS system, the factor can be used in its uncomplemented form , or it can be used in its complemented form. But in the RSYN system [22, 23], the usage of a common factor is restricted such that it can be used only in its uncomplemented form but not in its complemented form. Let F be a common factor and y be the variable representing the factor. The variable y must appear in positive phase only (i.e., it must appear as y but not as $\overline{y}$) inside the function which is represented in sum-of-products form of all the nodes which share that factor. In other words, a factor can be used only in its positive phase and not in its negative phase. We propose the use of such a synthesis method to generate the multilevel circuit to implement the output logic.

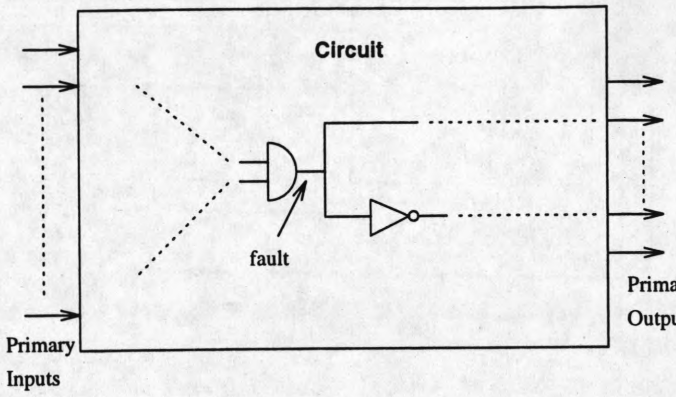

Figure 4: A fault in a general multilevel circuit can produce a bi-directional error at the primary outputs

Theorem 1 *The sequential circuit synthesized using the above design methodology is fault-secure with respect to all the faults except the faults at the primary inputs.*

Proof:

(1) `Faults in next state logic`

Let us assume there is a fault at the output of an internal node ν in the next state logic. A stuck-at fault on that node will be reflected as a single bit error at the state vectors. This will result in a noncode word at the state vector, and will be detected by the parity checker.

(2) `Faults in flip-flops`

Let us assume there is a stuck-at fault at a flip flop. A stuck-at fault on that node will be reflected as a single bit error at the state vectors. This will result in a noncode word at the state vector, and will be detected by the parity checker.

(3) `Faults in output logic`

Let us assume there is a fault at the output of an internal node ν in the output logic. Since that node appears only in the positive phase in the expressions of its fanout nodes in the sum-of-product expression, a stuck-at-1 fault at the output of ν will result in a unidirectional error (0 to 1) at its fanouts. This process will produce unidirectional errors (0 to 1) at the primary outputs of the circuit. By following a similar argument, we can show that a stuck-at-0 fault at the output of ν will result in unidirectional errors (1 to 0) at the primary outputs. Any fault inside the complex gate which implements the functionality of ν will affect the output of ν if the fault is sensitized. Thus, for some input combinations for which the fault is sensitized, the fault will look like either a stuck-at-1 or a stuck-at-0 fault at the output of ν. As shown before, this will result in unidirectional errors at the primary outputs of the circuit and will be detected by the Berger checker.

(4) `Faults in primary inputs`

The primary inputs may appear in both positive and negative phases in the expressions of primary outputs

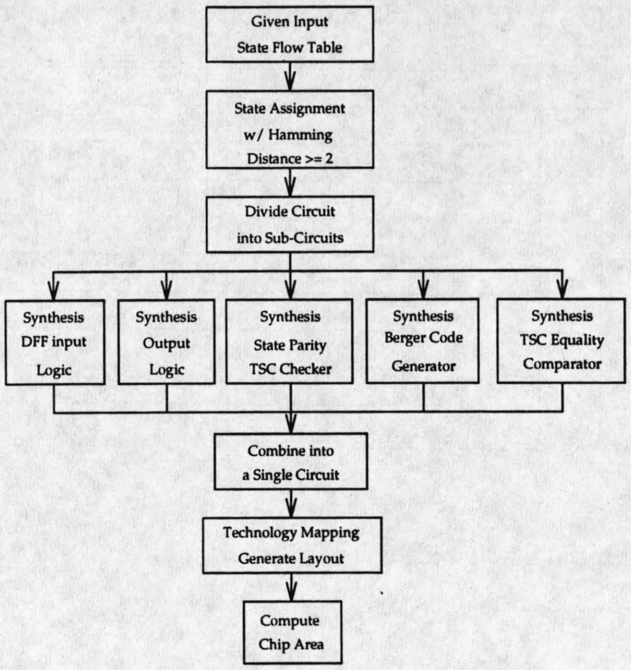
Given Input State Flow Table
State Assignment w/ Hamming Distance >= 2
Divide Circuit into Sub-Circuits
Synthesis DFF input Logic
Synthesis Output Logic
Synthesis State Parity TSC Checker
Synthesis Berger Code Generator
Synthesis TSC Equality Comparator
Combine into a Single Circuit
Technology Mapping Generate Layout
Compute Chip Area

Figure 5: Circuit synthesis flow chart.

in two-level form. Hence, potentially it can produce a 0 to 1 error at one primary output and a 1 to 0 error at another primary output. Thus, the faults at the primary inputs does not produce unidirectional errors at the primary outputs in general. These faults need to be checked using totally self checking dual rail input checkers. □

5.1 Implementation Issues

There are several implementation issues that need to be added to the proposed synthesis process: State assignment of the internal states, splitting up the state-generation into separate cones, and merging of all the cones, output logic, etc. The order of the implementations are shown in Figure 5.

5.2 State Assignment

For the sequential circuit to be capable of detecting single errors, the Hamming distance between any two states has to be at least two. In our implementation, the following assignment strategy is used. The first state to be encoded is assigned '0', and a binary counter is reset to 0. All of the following states to be encoded will compare to all previously determined states to ensure that the minimum distance with all of the existing states is at least two. The binary counter is incremented when this minimum distance is less than two. If the number of bits is not enough to represent the new state, the length of binary bit representation for states is incremented by 1. In addition, a '0' is appended to the end of every existing state, with the counter reset to

Table 1: Different Iterations in the State Assignment

State	I1	I2	I3	I4	I5	I6	I7	I8	I9
state 1	00	00	000	000	0000	0000	0000	0000	00000
state 2		11	110	110	1100	1100	1100	1100	11000
state 3			011	011	0110	0110	0110	0110	01100
state 4				101	1010	1010	1010	1010	10100
state 5					0011	0011	0011	0011	00110
state 6						0101	0101	0101	01010
state 7							1001	1001	10010
state 8								1111	11110
state 9									00011

'0', and the new state being all '0's followed by a '1'. An example of a state assignment for a circuit of nine states is the following.

5.3 Splitting into sub-circuits

The circuit is divided into three major sections, the state-generation logic, the checking logic for both state and output bits, and the output logic.

In order to avoid multiple-bit changes due to a single error in the state-generation logic, the logic of each individual flip-flop cannot allow sharing of gates. So instead of producing a single circuit for all of the flip-flops, every input cone has to be generated separately with its primary inputs and the corresponding state flip-flops. In this fashion each of the cones is synthesized and optimized. The checking logic components in the circuit consist of the parity checker, Berger-code generator, and the dual rail comparator. Each of them is individually built to fit the overall circuit. All of the components need to be optimized so that they occupy the least area possible. The output logic circuitry, however, cannot be optimized in the same fashion because it cannot allow both positive and negative phase factors to coexist. This is to ensure that the Berger code generated will detect all unidirectional errors correctly as described earlier.

The Berger code generator has been implemented using full adders and half adders based on the efficient modular design for a Berger code generator using a recursive procedure described in [8]. The Berger code generator is called a maximal length Berger code generator if $I = 2^k - 1$, where I is the number of information bits and k is the number of check bits. This kind generator can be implemented by using a set of full adders. An example of Berger code generator is shown in Figure 6 for $I = 7$ and $k = 3$. If $I \neq 2^k - 1$, the Berger code generator is implemented using full adders as well as half adder modules.

The Berger code generator and the totally self-checking equality comparator is generated automatically by our synthesis system.

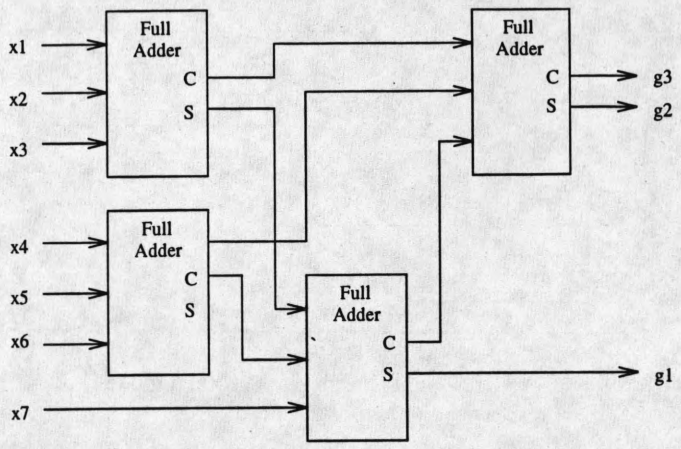

Figure 6: The Berger code generator for I=7, k=3.

The output network consists of three sections: a combinational logic section that generates the primary outputs and their corresponding encoded Berger code, a Berger code generator which produces the Berger check bits from the primary output bits, and a totally self-checking dual rail comparator that compares the encoded Berger bits with the generated Berger bits. Using this implementation, faults in the output circuit will be detected.

5.4 Merging of all sub-circuits

After all the sub-components of the circuit have been built, they need to be put together into a combined circuit that contains each of the flip-flop disjoint circuits, the non-inverting output circuit with Berger code, the parity checker for the state, Berger-code generator circuit, and the TSC dual rail checker.

Merging the logic driving disjoint flip-flops is trivial due to the fact that each of the flip-flop logic is already in its separate cone. The only conflict that may arise is that they may have identical internal node names. This can be avoided by appropriately renaming the internal nodes.

The task of merging the rest of the components involves more complexity. The inputs to these components have to be connected with outputs of other circuits. For instance, the inputs to the parity circuit are the outputs of the flip-flops; the inputs to the Berger-code generator are the primary outputs of the circuit, etc. Name-matching in this case is kept in a look-up table fashion so that the input names for each of these sub-circuits will correspond with the output names of the other circuits that feed them.

When merging of all the sub-circuits is completed, it is crucial that this combined circuit does not undergo any simplification or optimization, since by doing so the disjoint property of the input cones and the non-inversion property of the output circuit may be lost.

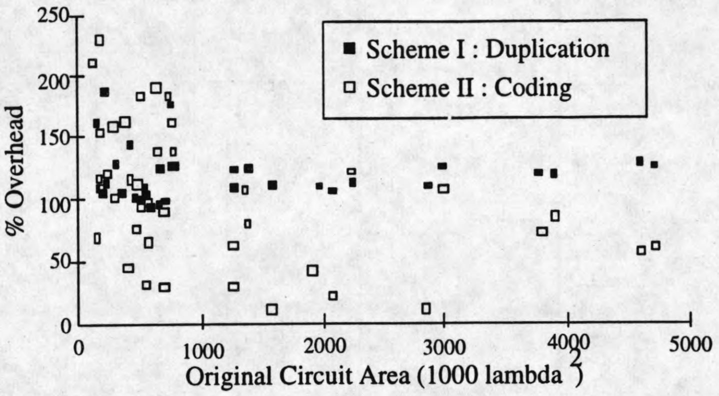

Figure 7: Plot of area overheads vs. original circuit size

6 Experiments and Analysis

The two synthesis methodologies described in the paper are used to compare the resulting layout areas. The first scheme uses straight-forward duplication of the original circuit with a TSC checker appended at the output to detect possible errors. The second scheme uses the coding scheme. We will now report on the results of the area overhead for the schemes for various sequential logic synthesis benchmarks. Each of the circuits were synthesized using our modification to the OCT/VEM synthesis system and mapped onto the 2 μ CMOS cell libraries of the OCT/VEM system. They were subsequently placed and routed by using the Timberwolf place and route package within the OCT/VEM system.

We want to emphasize here the fact that *all the necessary checkers that are needed for different schemes are generated* and the area required to implement them is included in the data presented in the subsequent tables. The area of the circuit is given in λ^2, where λ is the minimum size in any technology, which for a 2 micron technology refers to 1 micron.

Forty benchmark circuits are used for the experiments. The resulting areas range widely from a small circuit of size 163800 λ^2 to a larger circuit of size 4797700 λ^2. Table 2 shows the characteristics of the benchmark circuits along with the area information without any fault detecting capabilities. Table 3 shows the resulting areas using the duplication scheme and the coding scheme. From the results in the tables, a corresponding plot is shown in Figure 7. In this study, the circuits whose original areas are under 1000000 λ^2 are considered small circuits, while the rest of the benchmark circuits are considered to be larger circuits.

In the duplication approach (Scheme 1), there is a lower bound of 100% for duplication of the original circuit, with extra overhead due to the TSC comparator at the output. For the larger circuits, the overhead averages to be 110%. In the coding approach (Scheme 2) the area overhead shows a decreasing trend as the

Table 2: Original Benchmark Circuit Information and Layout Areas

Circuit Name	# Inputs	# Outputs	# States	Original Area
train4	2	1	4	163.8
tav	4	4	4	170.9
shiftreg	1	1	8	200.7
lion	2	1	4	201.2
mc	3	5	4	202.6
s27	4	1	6	248.7
bbtas	2	2	6	265.7
modulo12	1	1	12	328.8
beecount	3	4	7	344.3
dk27	1	2	7	383.2
dk15	3	5	4	391.8
dk17	2	3	8	434.4
train11	2	1	11	459.0
lion9	2	1	9	487.3
dk512	1	3	15	513.4
ex5	2	2	9	513.9
ex4	6	9	14	522.6
s8	4	1	5	589.7
opus	5	6	10	629.0
ex6	5	8	8	629.6
ex3	2	2	10	634.2
bbara	4	2	10	654.8
mark1	5	16	15	705.2
ex7	2	2	10	714.5
bbsse	7	7	16	804.1
sse	7	7	16	829.2
dk14	3	5	7	1195.1
ex2	2	2	19	1198.6
keyb	7	2	19	1371.1
cse	7	7	16	1400.6
s208	11	2	18	1588.6
s1a	8	6	20	1962.0
donfile	2	1	24	2101.8
s1	8	6	20	2158.4
dk16	2	3	27	2900.5
ex1	9	19	20	3010.9
sand	11	9	32	3787.3
styr	9	10	30	3895.6
planet1	7	19	48	4715.0
planet	7	19	48	4797.7

Note1: Areas are in lambda2.

Note2. Table is sorted by original areas.

Table 3: Area Overheads of Two Schemes

Circuit	Scheme I		Scheme II	
Name	Area	% Overhead	Area	% Overhead
train4	347.6	112.2	276.6	68.9
tav	440.0	157.5	520.5	204.6
shiftreg	421.4	109.7	442.2	120.3
lion	422.4	109.9	423.2	110.3
mc	570.7	181.7	660.0	225.7
s27	517.4	108.0	623.3	150.6
bbtas	565.0	112.6	585.6	120.4
modulo12	677.6	106.1	674.2	105.0
beecount	786.8	128.5	875.1	154.2
dk27	800.0	108.8	542.8	41.6
dk15	949.1	142.2	1014.5	158.9
dk17	929.9	114.1	941.9	116.8
train11	938.0	104.3	980.8	113.7
lion9	994.6	104.1	907.3	96.2
dk512	1087.9	111.9	989.1	92.7
ex5	1061.4	106.5	848.3	65.1
ex4	1284.9	145.9	1423.8	172.4
s8	1199.4	103.4	734.4	24.5
opus	1391.2	121.2	1462.3	132.5
ex6	1453.5	130.9	1746.1	177.3
ex3	1302.0	105.3	991.0	56.3
bbara	1343.2	105.1	1203.2	83.8
mark1	1893.8	168.5	1920.0	172.3
ex7	1462.6	104.7	871.1	21.9
bbsse	1799.7	123.8	2018.4	151.0
sse	1849.9	123.1	1924.9	132.1
dk14	2555.7	113.8	1456.2	21.8
ex2	2430.8	102.8	1822.8	52.1
keyb	2775.8	102.5	2769.1	102.0
cse	2992.7	113.7	2355.5	68.2
s208	3209.6	102.0	1657.4	4.33
s1a	4057.2	106.8	2619.7	33.5
donfile	4223.6	101.0	2405.0	14.4
s1	4450.0	106.2	4748.9	120.0
dk16	5862.1	102.1	3062.8	5.6
ex1	6588.4	118.8	5821.6	93.4
sand	7813.6	106.3	6642.9	75.4
styr	8027.3	106.1	7317.5	87.8
planet1	9996.6	112.0	7417.0	57.3
planet	10162.0	111.8	7655.2	59.6

Note1: Areas are in lambda2.

size of the circuit increases, as shown in Figure 7.

Clearly from the plot, it can be deduced that for very small circuits, it is often cheaper to implement high reliability by the simple duplication approach. On the contrary, as the circuit size grows larger, the coding scheme becomes much more attractive. This is intuitively so because for small circuits, the extra checking components added in the proposed scheme are comparable to the original circuit in size, in addition to the extra overhead of having disjoint cones for the input logic. For larger circuits, on the other hand, the overhead due to the extra logic is small compared to the original circuit, thus the proposed scheme out-performs the simple duplication approach.

7 Conclusions

In this paper, we proposed two schemes for reliability driven sequential logic synthesis. The first uses a straight-forward duplication approach, where the given circuit of the finite state machine is duplicated and the outputs of the machines are checked using a TSC comparator. The second scheme uses a coding approach. The states of the FSM are encoded using a distance 2 even parity code. The next state logic is implemented using independent logic cones for each state variable. The output logic is implemented using multilevel logic whose output is Berger coded, and is checked using Berger checkers.

The two schemes for synthesis have been integrated into the OCT/VEM synthesis tools. Experimental results on several sequential benchmark circuits have been obtained. The results show that for small circuits, the area overhead is smaller for the duplication scheme, namely about 150% compared to 200% for the coding scheme), while for large circuits, the area overheads for the coding sheme are quite small, about 60%, compared to the 110% for duplication schemes. Hence, we believe the coding scheme to be a very viable approach for reliable circuit synthesis for sequential circuits.

Future work in the area needs to be done on improving the coding schemes to come up with alternate coding methods. It would also be interesting to perform some research into improved state assignments using the distance 2 encoding which would be targeted to minimizing the area of the next state and output logic.

References

[1] B. W. Johnson, *Design and Analysis of Fault Tolerant Systems.* Addison Wesley, 1990.

[2] J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications.* North-Holland, 1982.

[3] R. Brayton, R. Ruddel, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A Multiple-level Logic Optimization System," *IEEE Trans. on CAD*, November 1987, pp. 1062-1081.

[4] R. Leveugle, R. Rochet, G. Saucier, "A Synthesis Tool for Fault-Tolerant Finite State Machines," *To appear in Fault Tolerant Computing Symposium*, June 1993.

[5] K. A. Barlett, D. Bostick, G. Hachtel, R. Jacoby, and M. Lightner, " BOLD: A Multiple-level Logic Optimization System," *International Conference on Computer Aided Design*, 1987.

[6] N. Weiner and A. Sangiovanni-Vincentelli, "Timing Analysis In Logic Synthesis," *26th Design Automation Conference*, 1989, pp. 655-661.

[7] K. De and P. Banerjee, "Logic Partitioning and Tesynthesis for Testability," *International Test Conference*, 1991.

[8] M. A. Marouf and D. A. Friedman, "Design of Self-Checking Checkers for Berger Codes," *IEEE 8th Annual Conference on Fault-Tolerant Computing*, June 1978, pp. 184-197.

[9] R. Leveugle, L. Martinez, "Design Methodology of FSMs with Intrinsic Fault Tolerance and Recovery Capabilities", EuroASIC'92, 1992, pp. 201-206.

[10] R. Leveugle, G. Saucier, "Optimized Synthesis of Concurrently Checked Controllers", *IEEE Trans. on Computers, vol. 39, no. 4*, April 1990, pp. 419-425.

[11] D. B. Armstrong, "A General Method of Applying Error Correction to Synchronous Digital Systems," *The Bell System Technical Journal*, vol. 40, no. 2, March 1961, pp. 577-593.

[12] R. L. Russo, "Synthesis of Error-Tolerant Counters Using Minimum Distance Tree State Assignments," *IEEE Trans. on Electronic Computers*, vol. EC-14, June 1965, pp. 359-366.

[13] J. F. Meyer, "Fault-Tolerant Sequential Machines," *IEEE Trans. on Computers*, vol. C-20, no. 10, October 1971, pp. 1167-1177.

[14] A. Sengupta, D. K. Chattopadhyay, A. Palit, A. K. Bandyopadhyay, M. S. Basu, A. K. Choudhury, "Realization of Fault-Tolerant and Fail-Safe Sequential Machines," *IEEE Trans. on Computers*, vol. C-26, no. 1, January 1977, pp. 91-96.

[15] A. Sengupta, D. K. Chattopadhyay, A. Palit, A. K. Bandyopadhyay, A. K. Choudhury, "Realization of Fault-Tolerant Machines-Linear Code Application," *IEEE Trans. on Computers*, vol. C-30, no. 3, March 1981, pp. 237-240.

[16] T. Villa, A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-level Logic Implementations", 26th DAC, 1989, pp. 327-332.

[17] S. Devadas et al., "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations", IEEE Trans. on CAD, vol. 7, no. 12, December 1988, pp. 1290-1300.

[18] P. K. Lala, *Fault-Testable Hardware Design.* Prentice-Hall International, London, 1985.

[19] V. P. Nelson, B. D. Carrol, *Tutorial Fault-Tolerant Computing.* IEEE Computer Society Press, Washington, D.C., 1986.

[20] S. Devadas, H. T. Ma, A. R. Newton, A. Sangiovanni-Vincentelli, "A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines," *IEEE Trans. on CAD*, vol. 8, no. 10, October 1989, pp. 1100-1107.

[21] N. K. Jha, and S. J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits and Systems," *International Conference on Computer Design*, 1991, pp. 578-581.

[22] K. De, C. Wu, and P. Banerjee, "Reliability Driven Logic Synthesis of Multilevel Circuits," *International Symposium on Circuits and Systems*, May 1992, pp. 1105-1108.

[23] K. De, C. Natarajan, D. Nair, P. Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. on VLSI Systems*, (submitted 1992).