Synthesis of Circuits with Low-Cost Concurrent Error Detection Based on Bose-Lin Codes

Debaleena Das and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
E-Mail: {ddas, touba}@ece.utexas.edu

Abstract

This paper presents a procedure for synthesizing sequential machines with concurrent error detection based on Bose-Lin codes. Bose-Lin codes are an efficient solution for providing concurrent error detection as they are separable codes and have a fixed number of check bits, independent of the number of information bits. Furthermore, Bose-Lin code checkers have a simple structure as they are based on modulo operations. Procedures are described for synthesizing circuits in a way that their structure ensures that all single-point faults can only cause errors that are detected by a Bose-Lin code. This paper presents an efficient scheme for concurrent error detection in sequential circuits with no constraint on the state encoding. Concurrent error detection for both the state bits and the output bits is based on a Bose-Lin code and their checking is combined such that one checker suffices. Results indicate low area overhead. The cost of concurrent error detection is reduced significantly compared to other methods based on other codes.

1. Introduction

In applications where dependability and data integrity are important, concurrent error detection (CED) circuitry is used to detect transient and intermittent errors. Early detection of errors is crucial for preserving the state of the system and preventing data corruption. The move towards deep-submicron technologies with lower voltage levels and smaller noise margins is increasing the susceptibility of systems to transient and intermittent faults thereby making CED increasingly important. This paper presents automated procedures for synthesizing both combinational and sequential circuits with low-cost CED circuitry. The CED circuitry can detect all on-line errors due to single-point faults as well as enhance off-line testability and reduce BIST overhead [1], [2].

One general approach for CED is to encode the outputs of a circuit with an error detecting code and have a checker that monitors the outputs and gives an error indication if a non-codeword occurs (as illustrated in Fig. 1).

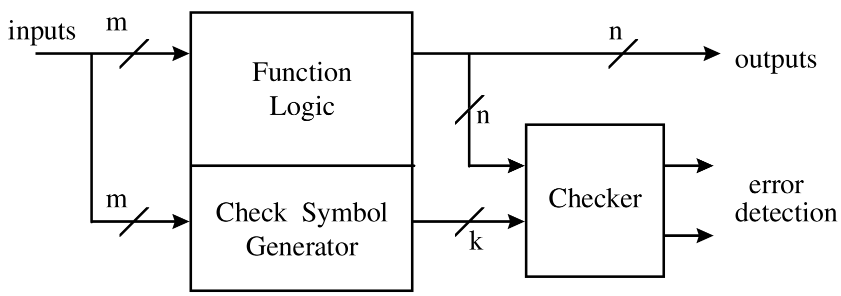

Figure 1. General Structure of a Self-Checking Circuit Based on Systematic Codes

To detect all single-point faults, the error detecting code that is used must be such that all possible errors in the circuit due to single-point faults result in non-codeword outputs. Efficient CED schemes exist for circuits with regular structures such as PLA's [3], [4] and ALU's [5], [6], [7]. However, for arbitrary multilevel combinational and sequential logic circuits more efficient approaches for synthesizing circuits with CED circuitry are needed [8].

One approach that has been proposed for synthesizing arbitrary multilevel circuits with concurrent error detection is based on a Berger code. A Berger code is an optimal code for detecting all unidirectional errors [9]. Jha and Wang [10] proposed a synthesis method in which the circuit is synthesized using only algebraic transformations (such as those used in the algebraic script in MIS

[11]) such that the resulting circuit can be transformed so that it is *inverter-free*, i.e., has inverters only at the primary inputs (PI's). The primary outputs (PO's) are then encoded with a Berger code. Since the inverters are only at the PI's, any error caused by an internal fault (i.e., fault not at the PI's) will produce a unidirectional error at the POs and therefore is guaranteed to be detected by the Berger code. Note that for any concurrent error detection scheme (including duplication), detection of faults at the primary inputs cannot be guaranteed unless encoded inputs are used. However, if the inputs to the circuit are outputs of another concurrently checked logic block, then the only undetectable PI faults are break faults after the checker [12]. De, *et al*., [13], described a constrained technology mapping procedure that maintains the inverter-free property during technology mapping thereby fully automating the synthesis method based on a Berger code. In [13], results are presented showing that a significant reduction in overhead is achieved by using the Berger code scheme compared with using duplication.

A Bose-Lin code [14] is an optimal systematic code for detecting up to t unidirectional errors and requires fewer check bits and a much simpler checker than a Berger code. In [7], it was shown that a significant hardware reduction could be achieved by using a Bose-Lin code instead of a Berger code for designing an ALU with CED. In this paper, a method is presented for synthesizing arbitrary multilevel circuits with CED based on a Bose-Lin code. The circuit is synthesized in a way that its structure ensures that all internal faults can only cause errors that are detected by a Bose-Lin code. Thus, a simpler Bose-Lin code can be used in place of a Berger code resulting in significant hardware reduction without loss of error detection capability.

Another contribution of this paper is a new and very efficient scheme for CED in sequential circuits. A sequential circuit with CED was proposed in [10]. The state bits are encoded with an m-out-of-n code and the output bits are encoded with a Berger code. Separate checkers are required to check the state bits and the output bits. In this paper, both the state bits and the output bits are encoded with a Bose-Lin code. An efficient technique is used to combine the checking the state bits and the output bits together with a single checker thereby reducing overhead. Moreover, since a Bose-Lin code is a separable code, no constraints are placed on the state encoding. Thus, it is compatible with conventional state assignment tools and also applicable to sequential circuits where some of the flip-flops may be storing data. By using the proposed procedure instead of a duplicate-and-compare approach, the cost of CED is significantly reduced while still detecting all single-point faults. The paper is organized as follows: Section 2 briefly explains some definitions and terms. Section 3 explains Bose-Lin codes. Section 4 presents the synthesis procedure. Sections 5 and 6 give the results for two approaches of CED for sequential circuits. Section 7 concludes the paper.

2. Definitions and Terms

Definition 1: A circuit is *fault-secure* if and only if for every fault in a specified fault class, the circuit never produces an incorrect codeword output for any codeword input.

Definition 2: A circuit is *self-testing* if and only if for every fault in a specified fault class, the circuit produces a non-codeword output for at least one codeword input.

Definition 3: A circuit is *totally self-checking* (TSC) if it is both fault-secure and self-testing. [15].

Definition 4: A circuit is *code-disjoint* if it maps input codewords (noncodewords) into output codewords (noncodewords).

Definition 5: A circuit is a *totally self-checking checker* if it is fault-secure, self-testing and code-disjoint.

Definition 7: A circuit is *path fault secure* (PFS) if and only if for every fault in a specified fault class, error propagation down any set of structured paths from the fault site to the outputs will never produce an incorrect codeword output [16].

The totally self-checking (TSC) goal is to detect the first error that occurs due to any fault in a specified fault class [16]. One way to achieve the TSC goal is through a self-checking circuit. Smith and Metze [16] introduced the concept of path fault secure circuits and showed it to be a subclass of fault secure circuits. Whereas the fault secure property depends on the functional paths in the circuit that are sensitized by codeword inputs, the PFS property depends on the structural paths and is therefore independent of the input set that is applied. It was shown in [16] that PFS circuits that are checked by a TSC checker achieve the TSC goal regardless of the input patterns applied during normal operation of the circuit. This property is important since we no longer need to consider the operating input patterns in the design process. The synthesis procedure implemented in this paper ensures the path fault secure property.

3. Bose-Lin Codes

Bose-Lin codes are optimal codes for detecting up to t unidirectional errors. These codes were developed by Bose and Lin [14]. They are systematic and require a fixed number of check bits, independent of the number of information bits. These two properties make Bose-Lin codes an efficient solution for synthesizing arbitrary circuits with CED. Being a systematic code, the check bits are appended to the normal output bits and thus no decoding is needed to get the normal output bits. The fixed number of check bits makes Bose-Lin codes an ideal choice for big circuits with a large number of outputs.

The codes are constructed by counting the number of zeroes, similar to the Berger codes. The counts are then modified depending on t. For $t = 2$ and 3, the counts are performed modulo 4 and 8, resulting in 2 and 3 check bits, respectively. Bose-Lin codes with check bits greater than 3 are explained in detail in [7]. The synthesis technique proposed in this paper can be used to implement self-checking circuits for any t unidirectional error detecting Bose-Lin codes.

The Bose-Lin and Berger codes are implemented using the number of zeroes or the bitwise inversion of the number of ones *only*. The error detecting property of the codes would fail if the number of ones were used for the implementation of the check bits. Consider an example with the information bits as 110010. If we used the number of ones for implementing a double error detecting Bose-Lin code, the resultant codeword would be 110010**11**. Let a unidirectional error 1→0 occur in the first and last bit positions *i.e.* the first information bit and the second check bit. This would yield the codeword 010010**10** and the error cannot be detected.

However, if the logic for generating the check bits is implemented independently from the functional logic, which generates the information bits, then any single-point fault can only affect either the check bits or the normal output bits (information bits). For this case, the error detecting property of the code does not fail if we use the number of ones in the check bit generation. We have made use of this fact in one of our schemes for CED where we use the number of zeroes in the check bits for one portion of the logic and the number of ones for the remaining portion of the logic to eliminate all potential aliasing problems.

Another important result occurs due to the independent generation of the check bits and the information bits. A t-error detecting Bose-Lin code can now detect up to 2^t-1 errors in the information bits using t check bits. This is due to the fact that a codeword changes to another codeword iff the number of output bits with unidirectional error is exactly equal to 2^t. This is a very significant result. We are able to get high error detecting capability with a small number of check bits. We exploit this result in the CED scheme proposed in this paper. Our CED scheme uses lesser check bits than CED based on Berger code, yet the additional structural constraints imposed are minor.

A totally self-checking checker design for Bose-Lin codes has been proposed in [17]. This checker is based on the Berger code checker proposed in [18]. Modulo 2^r addition is implemented by discarding the appropriate MSBs. Table 1 gives the comparison of literal counts of a double error detecting Bose-Lin code checker [17] as compared to a Berger code checker [18].

Table 1. Literal Count Comparison of Bose-Lin Code Checker versus Berger Code Checker

# of Bits	Berger [18]	Bose-Lin [17]
8	94	65
16	199	133
24	303	221
32	408	290

4. Synthesis Procedure

The circuit synthesis must be done such that all internal faults can only cause errors that can be detected by a Bose-Lin code. There are two structural constraints on the circuit that, if satisfied, will guarantee detection of all internal faults with a Bose-Lin code. The first constraint is that the circuit must be inverter-free so that only unidirectional errors can occur (this constraint is the same as for a Berger code). The second constraint is that no non-PI node in the circuit can have a path to more than 2^t-1 PO's if a t unidirectional error detecting Bose-Lin code is used and the check bit logic and the functional logic is implemented separately. (If the check bit logic and the functional logic were not implemented separately, then no non-PI node in the circuit could have a path to more than t PO's.) These two constraints ensure that all errors due to internal faults can cause no more than 2^t-1 unidirectional errors and thus all errors will be detected by the Bose-Lin code.

Note that it is not guaranteed that the checkers will get all codewords necessary to test every checker fault. Missing codewords can lead to redundant faults in the checker. These missing codewords are don't care inputs to the checker. The synthesis procedure can remove the redundant faults based on the don't care information. However note that several modules in the checker (e.g. TRC, adder) may be optimized library elements. Custom synthesis to make the checkers self-testing would prevent the use of these library elements in the design.

Multilevel logic optimization improves circuit area by using operations that restructure and minimize the logic. As was described in [10], multilevel logic optimization with algebraic transformations (such as those used in the algebraic script in MIS [11]) will result in a circuit that can be transformed so that it has inverters only at the PI's. So if the synthesis is restricted to algebraic transformations, then the inverter-free constraint can be satisfied.

It has been shown in a recent work [19] that it is not necessary to localize all the inverters of the circuits at the primary inputs for errors to be unidirectional. Saposhnikov *et al* have proposed an efficient scheme for synthesis by which any arbitrary combinational circuit is transformed into an unidirectional circuit such that every single gate fault can only be propagated to the circuit outputs on paths with an even or an odd number of inverters. This scheme would yield better results than localizing all the inverters at the PIs. However, for the sake of comparison we have used the synthesis technique implemented in [10]. The synthesis technique in [19] would identically improve our results (for Bose-Lin code) and the results presented in [10] (for Berger code).

The constraint that no non-PI node can have a path to more than 2^t-1 PO's can be satisfied by restricting fanout in the circuit. Multilevel logic optimization operations that restructure the circuit must be constrained so that they do not introduce fanout that creates a path from some node to more than 2^t-1 PO's. There are two restructuring operations that introduce fanout, resubstitution and extraction [11]. When performing those two operations, a check must be made to ensure that any new fanout being introduced in the circuit does not violate the constraints.

In *resubstitution* one logic expression f is substituted into another logic expression g to reduce the literal count. Resubstitution introduces fanout in the circuit, as f will fanout to g. A check must be made to see if a particular resubstitution operation will create a path from some node to more than 2^t-1 PO's. If so, then the resubstitution operation is not permitted when synthesizing for a Bose-Lin code.

In *extraction*, a common subexpression is factored out from a set of logic expressions. Fanout is introduced from the subexpression to the logic expressions that it was factored out of. Again, a check must be made to see if a particular extraction operation will create a path from some node to more than 2^t-1 PO's. If so, then the extraction operation is not permitted.

We modified the MIS logic synthesis [11] system for use in synthesizing circuits with CED based on Bose-Lin Codes. MIS uses various filters to reduce the pairs of logic expressions that are considered for resubstitution (because there are so many of them). We added an additional filter to remove pairs of logic expressions for which adding fanout would create a path from some node to more than 2^t-1 PO's. We also modified the extraction routines in MIS so that they only factor out common subexpressions between logic expressions for which the fanout constraints are satisfied. We added a command to MIS that automatically adds the Bose-Lin check bit functions. This modified version of MIS can be used to synthesize circuits with CED based on any *t* unidirectional error detecting Bose-Lin code. The original circuit is simply read into MIS and the Bose-Lin check bit functions are added, and then the circuit is optimized using the algebraic script with the modified resubstitution and extraction commands.

The constrained synthesis procedure guarantees the *path fault secure* property. Provided a technology mapping procedure that preserves the structure of the circuit is used, e.g., tree-mapping [20], [21], all errors due to internal faults are guaranteed to be detected. Any internal single-point fault can propagate to either the check bits or to the information bits. Further, any such fault can propagate to at most 2^t-1 outputs. Therefore, no internal single stuck-at fault can cause the circuit to produce an incorrect codeword output. If a totally self-checking Bose-Lin code checker is used, the implementation that is generated by our synthesis procedure is *self-checking* for internal single-point faults.

5. Self-Checking Sequential Circuits : *m*-out-of-*n* State Encoding

Self-checking sequential circuits have been studied in [10]. The general form of a sequential circuit with CED as proposed in [10] is shown in Fig. 2. The state bits are encoded with an *m*-out-of-*n* code and the output bits are encoded with a Berger code. The state bits and the output bits are checked separately and the two checker outputs are then combined with a final two rail checker (TRC) module. Any single-point fault in the combinational part of the circuit can affect the outputs of the next state logic and/or the outputs of the output logic. The fault will be detected by the *m*-out-of-*n* checker and/or the Berger code checker respectively. However, faults in the flip-flops cannot be detected by this design.

Here we propose two approaches for cost-effective design of sequential circuits based on Bose-Lin codes. This section presents the first approach. This approach uses *m*-out-of-*n* code for state encoding as in [10], but the CED for the outputs is based on Bose-Lin codes. Results are compared with the results presented in [10]. The second approach is a more general one where no constraint is placed on the state encoding.

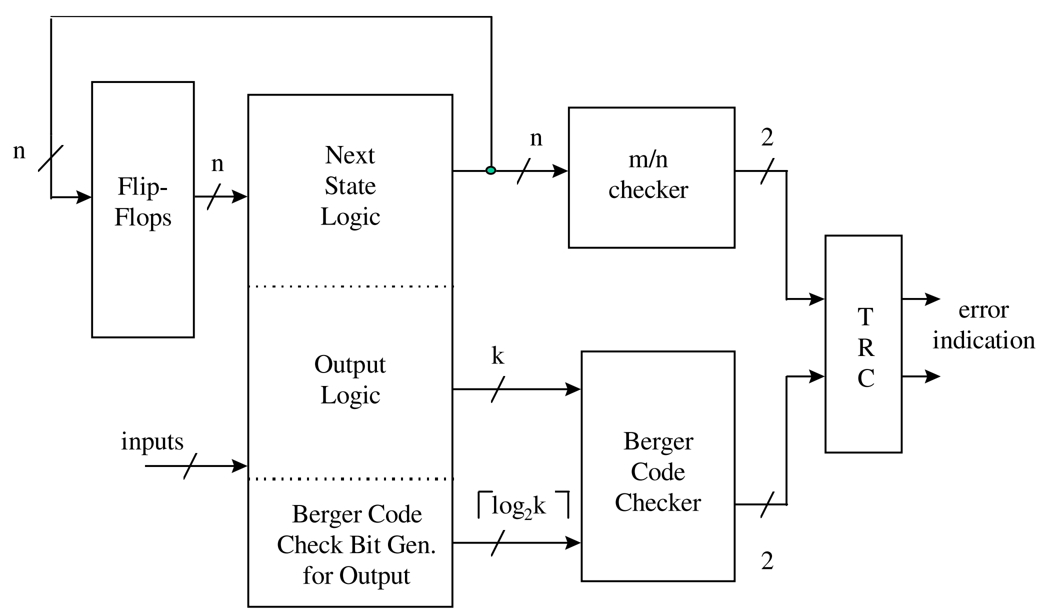

Figure 2. Self -Checking Sequential Machine Proposed in [10]

Our design for a self-checking sequential machine is illustrated in Figure 3. The state bits are encoded with an m-out-of-n code and the output bits are encoded with a double error detecting Bose-Lin code. To keep the hardware overhead low, the number of check bits should be kept to a minimum. t=2 is the smallest value which can detect all unidirectional faults. Parity (t=1) cannot detect all unidirectional faults. The cost of choosing a low t is an increase in the synthesis constraints. Increase in the synthesis constraints leads to an increase in the number of gates required. However, it also leads to reduced routing complexity and thus finally yields low layout areas. Since we used double error detecting codes, there are always only two check bits that are added to the output bits. The check bits indicate the number of output bits that have a logical value of '0' modulo 4. Synthesis is done as explained in Section 4 to ensure detection of all single-point faults by a double error detecting Bose-Lin code.

<u>Analysis of fault detection</u>: Our synthesis procedure ensures that a non-PI node does not have a path to more than three POs. Thus, a single-point fault can result in errors in the following bit combinations: three next state bits, three output bits, two next state bit and one output bit, one next state bit and two output bits, or a subset of any of the previous four. Any error in the next state bits is detected in the next clock cycle by the m-out-of-n checker checking the present state bits. Up to three errors in the output bits is detected by the Bose-Lin checker. Faults in the flip-flop are also detected by checking the present state bits rather than the next state bits.

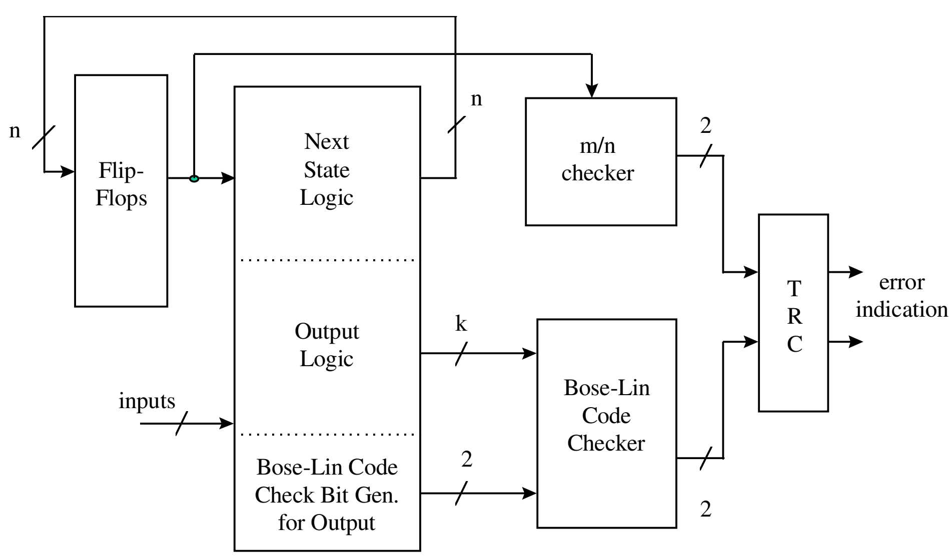

Figure 3. Self-Checking Sequential Machine With Output CED Based on Bose-Lin Codes

Table 2 gives the information about the benchmark circuits that were used for synthesizing sequential machines with CED. The circuits are sorted in increasing number of states. Table 3 gives the literal count comparison for the sequential machines with output CED based on Berger codes as reported in [10] and our sequential machines with output CED based on Bose-Lin codes. The total literal count includes the combinational logic, flip-flops and two checkers. State encoding is done using *m*-out-of-*n* codes, also known as *m*-hot codes. Smaller values of *m* yield circuits with lower literal-counts. However, the number of flip-flops required to encode the states may be too high. Results are presented only for that *m*-hot encoding which requires a feasible number of flip-flops (close to the minimal state assignment *i.e.* minimal number of flip-flops are used to encode the states). 2-hot encoding is used for *kirkman,* rather than 3-hot encoding as no results are available in [10] for 3-hot encoding for *kirkman.*

Table 2. Benchmark Sequential Circuits

Circuit	#inputs	#outputs	#states
dk15	3	5	4
mc	3	5	4
dk14	3	5	7
ex6	5	8	8
ex4	6	9	14
bbsse	7	7	16
cse	7	7	16
kirkman	12	6	16
sse	9	10	16
ex1	9	19	20
dk16	2	3	27
styr	9	10	30
sand	11	9	32
planet	7	19	48
scf	27	56	121

It can be seen in Table 3 that the number of literals for CED based on Bose-Lin codes is significantly reduced compared to the Berger code case. One advantage of Bose-Lin code lies in a fixed number of check bits, independent of the number of outputs of the original circuit. Thus for *scf* the double error detecting Bose-Lin code would require 2 check bits as compared to 6 for Berger code. This is reflected in columns 3 and 6 that give the literal counts for the combinational logic part of the machine. Another major advantage of Bose-Lin codes is the simpler checker. Due to the added advantage of a simpler checker, CED with Bose-Lin codes is seen to yield *much* better total literal counts than CED with Berger codes.

The percentage overheads of duplication, CED with *m*-hot encoding + Berger code [10] and our CED scheme with *m*-hot encoding + Bose-Lin code are presented in Table 4. Percentage overhead is computed as {[(literal count of circuit with CED) - (literal count of original circuit with minimal state assignment)] / (literal count of original circuit with minimal state assignment)} x 100. The second column in Table 4 gives the number of literals with no CED and state encoding with MUSTANG [22], which does state assignment with minimal number of flip-flops. This is taken as the literal count for the original circuit. Data for duplication is taken from [10]. Data for the CED schemes is taken from Table 3.

Table 4 shows that Bose-Lin code yields better results than other schemes for CED. Bose-Lin codes are most effective for circuits with a large number of outputs: *ex1* (19 POs), *planet* (19 POs) and *scf* (56 POs). Further, the results indicate that CED based on Bose-Lin codes is a practical solution for reasonably large circuits.

Table 3. Literal Count Comparison for Berger versus Bose-Lin Code

		Berger Code			Bose-Lin Code		
Circuit	*m*-hot	Comb. With CED	Checker + FF	Total	Comb. With CED	Checker + FF	Total
dk15	2	169	61	**230**	128	47	**175**
mc	2	67	68	**135**	58	54	**112**
dk14	2	235	81	**316**	152	67	**219**
ex6	2	173	125	**298**	154	97	**251**
ex4	3	165	140	**305**	137	113	**250**
bbsse	3	241	108	**349**	185	88	**273**
cse	3	431	108	**539**	304	88	**392**
kirkman	2	467	121	**588**	323	100	**423**
sse	3	241	108	**349**	210	74	**284**
ex1	3	561	256	**817**	448	198	**646**
dk16	3	457	94	**551**	451	94	**545**
styr	3	952	182	**1134**	652	148	**800**
sand	3	806	170	**976**	712	143	**855**
planet	4	810	314	**1124**	741	256	**970**
scf	4	1373	862	**2235**	1141	688	**1829**

Table 4. Percentage Overhead Comparison of CED Schemes

Circuit	Original	Duplication	Berger	Bose-Lin
Dk15	68	188	238	157
Mc	24	350	462	366
Dk14	93	180	240	135
Ex6	99	199	201	153
Ex4	76	258	301	228
Bbsse	140	174	149	95
Cse	231	145	133	70
Kirkman	210	146	180	101
Sse	122	185	186	133
Ex1	269	180	203	140
Dk16	283	130	95	93
Styr	489	129	132	64
Sand	553	124	77	56
Planet	575	207	96	69
Scf	971	155	130	88

6. Self-Checking Sequential Circuits : Unconstrained State Encoding

The CED scheme for sequential machines proposed in [10] uses a specific state encoding scheme, namely state encoding using *m*-hot codes. It should be noted here that it is not *always possible* to encode the states using *m*-hot codes. In this section we propose a scheme for CED in sequential machines where Bose-Lin codes are used to encode both the state bits and the output bits. A clever scheme is used to combine the checking of the state bits and the output bits such that one checker suffices. Bose-Lin codes are separable codes and thus no constraints are placed on the state encoding. Thus, this scheme of CED is compatible with MUSTANG and any other state assignment tool. Also, it is applicable to general sequential circuits where some of the flip-flops may be storing data. This scheme can be used to convert an existing sequential circuit (with both control state *and* data flip-flops) that does not have CED into one with CED by simply eliminating a sufficient number of fanouts to guarantee that all errors due to internal faults will be detected.

Double error detecting Bose-Lin codes are used to encode both the state bits and the output bits. An efficient technique is used to combine the checking of the state bits and the output bits using a single checker (as illustrated in Fig. 4). We have used the following naming convention: *NS* (Next State bits) denotes the state bits generated from the next state logic block. *PS* (Present State bits) denotes the state bits after the flip-flops *i.e.* they are the *NS* after one clock cycle. NS_C are the two check bits for the next state logic. After passing through the flip-flop stage, they become check bits for the present state and are denoted as PS_C. Finally, Z_C are the check bits that encode both the output logic as well as the *PS*.

NS_C bits are computed in the usual manner by computing the number of ones modulo 4 in *NS*. Recall that if the check bit generator is implemented separately from the original logic, the number of ones or the number of zeroes can be used for encoding. The check bits for the output logic (Z_C), however, are computed as the number of ***zeroes** modulo 4 in the output bits* ***plus*** *the number of* ***ones*** *modulo 4 in PS.* We have used the number of ones for the state bits and number of zeroes for the output bits to avoid aliasing. This is explained further in the subsection on analysis of fault detection. The check bits, Z_C, can be thought of as encoding both the input and the output space of the circuit. Thus, only one checker is sufficient to check both the spaces. The Bose-Lin code checker has to be slightly modified for this application. The output bits are fed to the checker and the number of zeroes modulo 4 is computed. PS_C is added to the result of this computation and then the result is compared with Z_C. This arrangement ensures that the state bits and the output bits are checked together and all faults in the flip-flops are also detected.

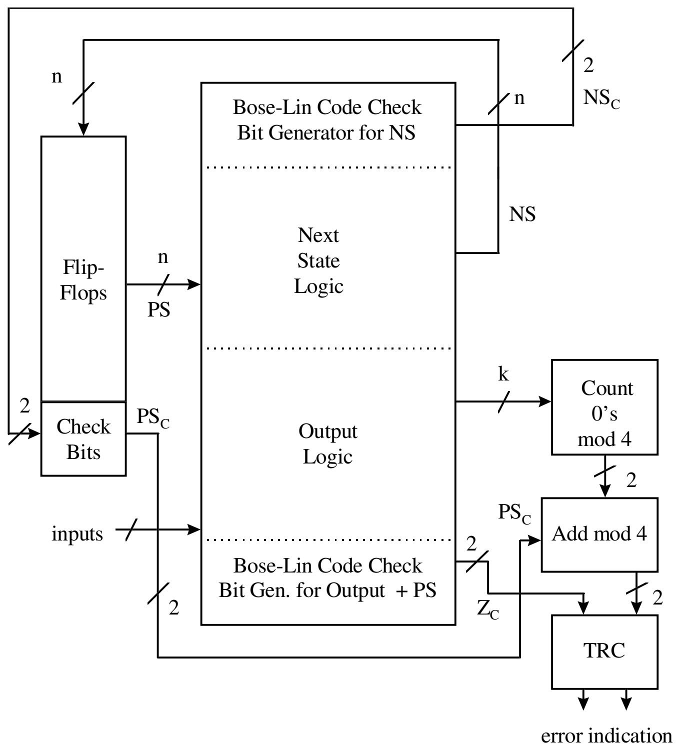

Figure 4. Proposed Self-Checking Sequential Machine With CED Based on Bose-Lin Code

<u>Analysis of fault detection</u>: Our synthesis technique ensures that a non-PI node does not have a path to more than three POs. Thus, a single-point fault can result in errors in the following bit combinations: three *NS* bits, three output bits, two *NS* bit and one output bit, one *NS* bit and two output bits, or a subset of any of the previous four. Errors in the *NS* bits will be clocked into the flip-flops. In the next cycle they are propagated to the *PS* bits and detected by the checker. Errors in the output bits will be detected in the same clock cycle as the occurrence of the error. Note that faults in the flip-flops are also detected since the checker checks the *PS* rather than *NS*. Even though the flip-flops may fanout to more than three PO's, single errors in the flip-flops are equivalent to being in a non-codeword state which is guaranteed to cause a mismatch with the PS_C check bits.

Consider the possible error combinations in the output bits. There is a possibility of aliasing only if a fault occurs on a node which fans out to both the next state and the output bits. Without loss of generality assume a stuck-at-0 fault on this node. Recall that our synthesis procedure ensures that any single-point fault in the circuit can only cause unidirectional errors at the output. Consider the case where the fault propagates to one of the next state bits in *clock cycle* - *clk1* and to one of the output bits in the next consecutive *clock cycle* - *clk2*. Say, we had used the number of ones in the computation of NS_c as well as Z_c. Then NS_c in *clk1* and PS_c in *clk2* would be one more than the number of ones in *NS* and *PS* respectively since one of the next state bits is erroneously set to 0 in *clk1*. Further, Z_c in *clk2* would also be one more than the number of ones in *PS* and output bits since one of the output bits is erroneously set to 0 in *clk2*. Thus, the errors would have cancelled out. However, we have completely eliminated the possibility of any such aliasing by using the number of ones to encode the next state bits and the number of zeroes for the output bits. A stuck-at-0 fault would cause PS_c to be one *more* than the actual number of ones in the state bits and Z_c to be one *less* than the number of ones in the present state bits plus the number of zeroes in the output bits.

Table 5 gives the comparison of the proposed method with [10]. State assignment has been done using MUSTANG [22]. The number of flip-flops for the smaller circuits is the minimal number required to represent the states. Values of *m* for the *m*-hot encoding have been taken such that the number of flip-flops is comparable in both schemes.

Table 5 shows that the number of literals in the circuit with CED based on Bose-Lin codes is less for small circuits and larger for bigger circuits as compared to CED based on Berger codes. This is due mostly to the fact that an *m*-hot encoding gives better literal count results than MUSTANG for machines with a large number of states [10]. In actuality, comparing our literal counts with those in [10] is not a good comparison because the state assignment in the two cases is so different. We show the comparison just to illustrate that even with a less optimal state assignment (from the perspective of minimizing the literal count), we still get better results in most cases.

Table 5 also indicates that despite a larger number of literals in the circuit with CED, the total number of literals is comparable for both methods. The saving in cost of CED results from the smaller number of check bits for the Bose-Lin codes and the other saving in literals by our checker scheme. The modification to the normal Bose-Lin checker in our sequential circuit CED scheme is the modulo 4 addition of PS_c to the number of zeroes modulo 4 of the output bits. This requires 14 literals. Since the Bose-Lin code checker requires fewer literals than the Berger code checker, and we are using only one checker, the total number of literals in our proposed checker is much less than that of Jha and Wang [10] for a reasonably large number of outputs.

Layout was done for the circuits in Table 5. We felt that our synthesis constraint that no internal node can fanout to more than two primary outputs would reduce routing complexity and thus lead to less layout area even when the number of literals is comparable. This is indeed the case as is seen in Table 6. Since layout areas were not given in [10], we computed the layout areas for that scheme too. Layout was targeted for standard cell 2 micron ORBIT process (*MDK* library) of Mentor Graphics.

The layout areas are in general lesser for minimal state assignment and CED for the output bits based on Bose-Lin codes. *dk16* shows better result for the *m*-hot assignment and CED for the output bits based on Berger code. This is an expected result since *dk16* has only 3 outputs and so there is no benefit of using Bose-Lin code. *m*-hot assignment yields a compact layout that balances out the saving from the combining of the checkers in our scheme. The percentage overheads from Table 6 indicate that our CED scheme, which is applicable for any general sequential circuit and is compatible with any state assignment tool, is a practical solution.

Table 5. Literal Count Comparison for Berger Code with *m*-hot state assignment versus Bose-Lin Code with Minimal State Assignment

	m-hot Assign + Berger Code			Min. State Assign. + Bose-Lin		
Circuit	Comb. With CED	Checker + FF	Total	Comb. With CED	Checker + FF	Total
Dk14	235	81	**316**	204	77	**281**
Dk15	169	61	**230**	122	74	**196**
Dk16	457	94	**551**	496	63	**559**
Planet	810	314	**1124**	883	226	**1109**
Styr	952	182	**1134**	1120	129	**1249**

Table 6. Layout Area Comparisons

Circuit	Original	*m*-hot Assign +Berger Code	% Ovhd.	Min. state Assign. + Bose-Lin Code	% Ovhd.
dk14	1148560	1920000	67	1635300	42
dk15	618750	1595000	158	1127000	82
dk16	2593500	3675000	42	3937500	52
Planet	7425950	12624115	70	12059500	62
Styr	5587000	9050940	62	8855000	59

7. Conclusions

It has been found that Bose-Lin codes significantly reduce the cost of concurrent error detection. Synthesis and layout of benchmark sequential circuits with CED based on Bose-Lin codes show low hardware overheads, making it a practical solution. Results obtained show a significant reduction in overhead as compared to CED based on Berger code (which has been shown to yield the best results so far). An efficient scheme for CED in sequential circuits has been proposed. This scheme can be extended to handle CED based on Berger codes. The advantages of this scheme are the following:

1) Only one checker is required for checking both the state bits and the output bits.
2) No constraints are placed on state encoding.
3) Faults in the flip-flops are guaranteed to be detected.

The synthesis procedure outlined in this paper was implemented in MIS, but can be implemented in other logic synthesis systems. Note also that the scheme described in this paper can be used to convert an existing sequential circuit (with both control state *and* data flip-flops) that does not have CED into one with CED by simply eliminating a sufficient number of fanouts to guarantee that all errors due to internal faults will be detected.

References

[1] M. Nicolaidis, "Self-Exercising Checkers for Unified Built-In Self-Test (UBIST)," *IEEE Trans. on Computer Aided-Design*, Vol. 8, No. 3, pp. 203-218, Mar. 1989.

[2] S.K. Gupta, and D.K. Pradhan, "Can Concurrent Checkers help BIST?," *Proc. IEEE Int. Test Conf.,* pp. 140-150, Sept. 1992.

[3] G.P. Mak, J.A. Abraham, and E.S. Davidson, "The Design of PLAs with Concurrent Error Detection," *Proc. FTCS,* pp. 303-310, Jun. 1982.

[4] M. Nicolaidis, and M. Boudjit, "New Implmentations, Tools, and Experiments for Decreasing Self-Checking PLAs Area Overhead," *Proc. of International Conference on Computer Design*, pp. 275-281, Oct. 1991.

[5] D.K. Pradhan, *Fault Tolerant Computing: Theory and Techniques,* Vol. 1, Englewood Cliffs, NJ: Prentice-Hall, 1986, Chap. 5.

[6] J.-C. Lo, S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. on Computer Aided-Design*, Vol. 11, No. 4, pp. 525-540, Apr. 1992.

[7] S. Gorshe, and B. Bose, "A Self-Checking ALU Design with Efficient Codes," *Proc. of VLSI Test Symposium*, pp. 157-161, Apr. 1996.

[8] M. Goessel, and S. Graf, *Error Detection circuits*, London, NY: McGraw-Hill, 1993.

[9] J.M. Berger, "A Note on Error Detecting Codes for Asymmetric Channels," *Information and Control*, Vol. 4, pp. 68-73, Mar. 1961.

[10] N.K. Jha, and S.Wang, "Design and Synthesis of Self-Checking VLSI Circuits," *IEEE Trans. Computer-Aided Design,* Vol. 12, No.6, pp. 878-887, Jun. 1993.

[11] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. on Computer Aided-Design*, Vol. 6, pp. 1062-1081, Nov. 1987.

[12] B. Khodadad-Mostashiry, "Break Faults in Circuits with Parity Prediction," *Technical Note No. 183,* Center for Reliable Computing, Stanford University, Stanford, CA, Dec. 1980.

[13] K. De, C. Natarajan, D. Nair, and P. Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. VLSI Systems,* pp. 186-195, Jun. 1994.

[14] B. Bose, and D.J. Lin, "Systematic Unidirectional Error-Detecting Codes," *IEEE Trans. on Computer*

Aided-Design, Vol. C-34, No. 11, pp. 1024-1032, Nov. 1985.

[15] D.A. Anderson, "Design of Self-Checking Digital Networks Using Coding Techniques," *Technical Report R-527*, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1971.

[16] J.E. Smith, and G. Metze, "Strongly Fault Secure Logic Networks," *IEEE Trans. Computers,* Vol. C-27, No.6, pp. 491-499, Jun. 1978.

[17] N.K. Jha, "Totally Self-Checking Checker Designs for Bose-Lin, Bose and Blaum Codes," *IEEE Trans. Computer-Aided Design,* Vol. 10, No. 1, pp. 136-143, Jan 1991.

[18] M.A. Marouf and A.D. Friedman, "Design of Self-Checking Checkers for Berger Codes," *Proc. FTCS,* pp. 179-184, Jun. 1978.

[19] V.V. Saposhnikov, A. Morosov, VL. V. Saposhnikov, and M. Goessel, "A New Design Method for Self-Checking Unidirectional Combinational Circuits," *JETTA,* Vol. 12, Issue ½, pp. 41-53, Feb. 1998.

[20] K. Keutzer, "Dagon: Technology Binding and Local Optimization by DAG Matching," *Proc. IEEE/ACM 24th Design Automation Conf.*, pp. 341-347, Jun. 1987.

[21] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology Mapping in MIS," *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, pp. 116-119, Nov. 1987.

[22] S. Devadas, Ma Hi-Keung, A.R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations," *IEEE Trans. on Computer Aided-Design*, Vol. 7, No. 12, pp. 1290-1299, Dec. 1988.