Approximate logic circuits for low overhead, non-intrusive concurrent error detection

Mihir R. Choudhury and Kartik Mohanram
Department of Electrical and Computer Engineering
Rice University, Houston, TX 77005
{mihir,kmram}@rice.edu

Abstract

This paper describes a scalable, technology-independent algorithm for the synthesis of approximate logic circuits. A low overhead, non-intrusive solution for concurrent error detection (CED) based on such circuits is described in this paper. CED based on approximate logic circuits does not impose any performance penalty on the original design. The proposed synthesis algorithm for approximate logic circuits scales with circuit size, and provides fine-grained trade-offs between area-power overhead and CED coverage.

1. Introduction

It is widely acknowledged that there will be a sharp increase in hardware failures in scaled CMOS technologies, e.g., [1, 2]. In addition to variability and manufacturing defects, single-event effects and new failure mechanisms such as transistor wear-out on speedpaths in logic circuits are projected to pose significant challenges to hardware reliability with technology scaling. As lifetime reliability and robustness, alongside performance and power, emerge as key factors that drive synthesis and design, there is significant interest in the development of cost-effective techniques for robust error-resilient design of logic circuits.

Circuits with concurrent error detection (CED) have the capability to detect both temporary and permanent faults and have historically been used in systems where dependability and data integrity are of importance. Conventional techniques for synthesis of CED for multi-level logic circuits focus on guaranteeing 100% coverage of broad classes of errors and generally require very large area, power, and performance overhead (often in excess of 100%) [3–9]. For high-volume mainstream applications, most of these existing CED solutions will be overkill. Recent research has seen the emergence of CED techniques that try to meet coverage requirements at minimum cost instead of trying to guarantee coverage of broad classes of errors, e.g., [10, 11]. However, these techniques have one or more disadvantages that limit their integration into standard design flows. These include limited scalability and options to trade-off coverage for overhead, requiring modifications to or constraining synthesis of the original design (intrusive CED), performance penalty due to longer critical path delay in CED logic, and technology-dependent synthesis.

This paper describes a scalable, technology-independent algorithm for the synthesis of a new class of circuits called approximate logic circuits. An approximate logic circuit is constructed by selectively converting minterms from the off-set or on-set of its Boolean function into don't cares. The intuition is that by converting only a small fraction of minterms into don't cares, an approximation of a Boolean function can be synthesized with a significantly lower area-power-delay footprint than the exact Boolean function. Since brute-force enumeration and conversion of minterms into don't cares does not scale with circuit complexity, an efficient algorithm that manipulates a technology-independent multi-level network to synthesize approximate logic circuits is described in this paper. The algorithm iteratively reduces the Boolean expressions of the nodes in the multi-level network using two cube selection techniques based on exact cubes and observability don't care cubes. Note that whereas approximation of Boolean functions has been studied in theoretical computer science, these works have no direct applications to synthesis of approximate logic circuits as proposed in this paper.

A low overhead, non-intrusive solution for CED based on approximate logic circuits is described in this paper. The proposed synthesis algorithm for approximate logic circuits provides fine-grained trade-offs between area-power overhead and CED coverage. Since CED is non-intrusive, no modifications are necessary to the original design. A self-checking checker that is compatible with the proposed CED technique is also described. The checker produces two-rail encoded outputs, ensuring compatibility with other error detection techniques for error signal consolidation. Simulation results for 8 benchmark circuits indicate that CED coverage of 81% can be achieved for 25% and 34% area and power overhead on average. CED based on approximate logic circuits incurs no performance penalty, since approximate logic circuits are significantly smaller than the original circuit. For the benchmarks studied in this paper, the delay of the approximate logic circuit was 38% less than the delay of the original circuit on average.

Finally, the synthesis algorithm is extended to include logic sharing between the original and approximate logic circuits to further reduce overhead. Results show that for the same coverage, CED using approximate logic circuits and logic sharing always requires a lower overhead in comparison to state-of-the-art intrusive techniques for CED.

This paper is organized as follows. Section 2 introduces approximate logic functions and describes the proposed synthesis algorithm. Section 3 describes the application of approximate logic circuits to CED. Section 4 presents simulation results. Section 5 is a conclusion.

2. Approximate logic functions

Given a Boolean function $\mathcal{F}$, a Boolean function $\mathcal{G}$ is a 0-approximation of $\mathcal{F}$ *iff* $\overline{\mathcal{G}} \Rightarrow \overline{\mathcal{F}}$. Similarly, $\mathcal{G}$ is a 1-approximation of $\mathcal{F}$ *iff* $\mathcal{G} \Rightarrow \mathcal{F}$. If all the input vectors are equally likely, the *approximation percentage* can be defined as the fraction of minterms that are covered by the approximate function in the 1(0)-minterm space of the exact function $\mathcal{F}$. If the input vectors are not equally likely, then each minterm that is covered by the approximate function must be

This research was supported in part by grant CCF-0701547 from the National Science Foundation and in part by a gift from Intel Corporation.
.

appropriately weighted by its probability of occurrence to compute the approximation percentage.

Approximate logic functions are interesting because a very high approximation percentage can be achieved with a low area overhead. This can be illustrated with the following example. Consider the function $\mathcal{F} = a + b + \overline{c}\overline{d} + cd$. The optimum implementation of the function with 1/2-input gates requires 7 gates. A 1-approximation of this function is given by the function $\mathcal{G} = a + b$. The implementation of $\mathcal{G}$ requires only 1 gate. Assuming that all input vectors are equally likely, $\mathcal{G}$ covers 12 out of 14 minterms in $\mathcal{F}$. Thus, an approximation percentage of 85.72% can be achieved for an area overhead of 1/7 or 14.28%. Since brute-force enumeration and conversion of minterms into don't cares does not scale with circuit complexity, an efficient algorithm that manipulates a technology-independent multi-level network to synthesize approximate logic circuits is described in the rest of this section.

2.1 Synthesis of approximate logic functions

The algorithm for approximate logic synthesis starts with a multi-level, technology-independent network of the original circuit [12]. The network is an interconnection of nodes in a directed acyclic graph where each node has two kinds of Boolean functions associated with it: (i) a function of its local inputs referred to as the local Boolean function of the node, and (ii) a function of the primary inputs to the circuit referred to as the global Boolean function of the node. The local Boolean function of nodes in the network can be expressed as a sum-of-products (SOP) expression in terms of the local fanin nodes. Every SOP can be written either in the zero phase (denoting the off-set) or in the one phase (denoting the on-set). A zero phase SOP can be converted to a one phase SOP and vice-versa using DeMorgan's law.

The approximate function is either a 0-implication or a 1-implication of the output of the original circuit. Although implication-based techniques for synthesis, verification, and test have been proposed in literature, e.g., [13], such techniques search for implication relations among Boolean functions implemented within the original circuit. In contrast, the synthesis algorithm proposed in this paper is technology-independent and searches for implications that may not be a part of the original circuit. The algorithm for synthesis of approximate logic functions is divided into 2 stages: (i) Type assignment: Assigning type of approximation (0/1/EX/DC) to each node in the multi-level network, and (ii) Cube selection: Reducing the SOPs of the nodes in the multi-level network by selecting a subset of cubes based on type assignment. The next two sections describe the two stages of the synthesis algorithm in detail.

2.1.1 Type assignment

The type assignment is done as a preprocessing step prior to cube selection. The aim of the type assignment algorithm is to determine the type of approximation (0/1/EX/DC) that must be made at each node in the technology-independent network to maximize the approximation percentage at the primary output.

Local observability values are used to assign a type to each node in the multi-level network. For each node g in the multi-level network, the local observability of the fanin nodes of g are computed with respect to the output of g. The *local 0(1)-observability of a fanin node* is defined as the probability that a 0(1) value at the fanin is observable at the output of the node. The intuition behind assigning a type to a node is that if the local 0(1)-observability is dominant, then a 0(1)-approximation of the node would yield a high approximation percentage at the output of the node. In order to leverage flexibility in type assignments, each node can be assigned one of 4 types: 0, 1, EX, or DC. Type 0(1) means that the 0(1)-minterm space of the global Boolean function at the node is essential to achieving a high approximation percentage at the output of the node. Type EX means that both 0 and 1-minterm spaces are essential. Finally, type DC means that neither 0 nor 1-minterm space is essential.

The type assignment algorithm is initialized by ascertaining the type of approximation desired at each primary output of the circuit. The processing of a node involves assigning a type to the node and then analyzing the local observabilities of the fanin nodes to *request* a type for each of its fanin nodes. The algorithm processes the nodes in the network in a reverse topological order, i.e., a node is assigned a type after all its fanout nodes have been assigned a type. Nodes are assigned types in such an order because the type assignment of a node depends on the types requested by its fanout nodes. The type assignment of a node is done based on the following rules:

- Any fanout node requests type EX, node is assigned type EX.
- All fanout nodes request type DC, node is assigned type DC.
- All fanout nodes request type 0/DC, node is assigned type 0.
- All fanout nodes request type 1/DC, node is assigned type 1.
- Else, node is assigned type EX.

After assigning a type to the node, a request for the type of each fanin node is made based on the local 0-observability and 1-observability of the fanin nodes. (i) If both local 0-observability and 1-observability of a fanin node are small as compared to the observabilities of other fanin nodes, a type DC is requested for the fanin node, (ii) if there is a big disparity between the local 0-observability and 1-observability of the fanin node (based on the ratio), then the type with the higher observability is requested for the fanin node, and (iii) if the local 0-observability and 1-observability are comparable, then a type EX is requested for the fanin node.

2.1.2 Cube selection

In the preprocessing stage described above each node was assigned a type to maximize the approximation percentage. Approximation of the Boolean function at the fanin of a node may cause the bit(s) in the input vector to the node to change. A (1)0-approximation may be rendered incorrect due to such bit flips if an input vector in the 0(1)-minterm space transforms into an input vector in the 1(0)-minterm space. The 1(0)-minterm space of a type 1 (type 0) node must be reduced to compensate for the bit flips and ensure correctness of approximation. This subset of the 1(0)-minterm space will be referred to as the feasible subspace of the node. Note that the feasible subspace of a node will be larger when lesser number of bit flips occur, i.e., when the approximation at the fanin nodes are close to the exact functions. Since the feasible subspace depends on the approximations at the fanin nodes, explicit computation of the feasible subspace is prohibitively complex.

The goal of cube selection is two fold: (i) to ensure correctness of approximation and (ii) to achieve a high approximation percentage for low overhead. Two techniques for cube selection, with computational complexity linear in the size of the technology-independent network, are described in this paper. The first technique, exact cube selection, guarantees the correctness of the approximate function by computing a subset of the feasible subspace. However, this technique may affect the approximation percentage because it imposes strict constraints for selecting cubes to guarantee correctness. The second technique, observability don't care based cube selection, relaxes the constraints for cube selection using local observability don't cares. However, this relaxation may result in the inclusion of minterms outside the feasible subspace, which does not guarantee

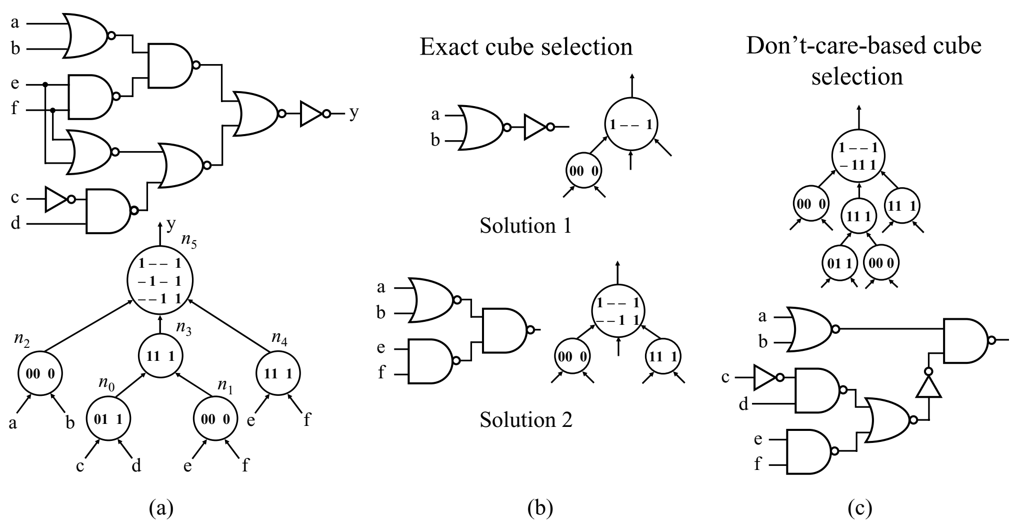


Figure 1: Example circuit (a) and approximate logic circuits for (a) using the two cube selection algorithms (b) and (c).

correctness. To ensure correctness while maintaining a high approximation percentage, an algorithm for cube selection that uses these two techniques iteratively is proposed.

Theorem: Given Boolean functions X_1, X_2, $\mathcal{F} = X_1X_2$, and 1-approximate Boolean functions X_1', X_2' for X_1 and X_2, then $\mathcal{F}' = X_1'X_2'$ is a 1-approximation for $\mathcal{F}$.

Proof: Since X_1', X_2' are 1-approximations for X_1, X_2, $X_1' \Rightarrow X_1$ and $X_2' \Rightarrow X_2$. Thus, $\overline{X_1'} + X_1 = 1$ and $\overline{X_2'} + X_2 = 1$

$$\begin{aligned}
&\Leftrightarrow\ (\overline{X_1'} + X_1)(\overline{X_2'} + X_2) = 1\\
&\Leftrightarrow\ \overline{X_1'}\,\overline{X_2'} + \overline{X_1'}X_2 + \overline{X_2'}X_1 + X_1X_2 = 1\\
&\Leftrightarrow\ \overline{X_1'}(\overline{X_2'} + X_2) + \overline{X_2'}(\overline{X_1'} + X_1) + X_1X_2 = 1\\
&\Leftrightarrow\ \overline{X_1'} + \overline{X_2'} + X_1X_2 = 1\\
&\Leftrightarrow\ \overline{X_1'X_2'} + X_1X_2 = 1\\
&\Leftrightarrow\ X_1'X_2' \Rightarrow X_1X_2\\
&\Leftrightarrow\ \mathcal{F}' \Rightarrow \mathcal{F}
\end{aligned}$$

In other words, $\mathcal{F}'$ is a 1-approximation of $\mathcal{F}$. Similarly, we can prove that if $\mathcal{F} = X_1+X_2$ then $\mathcal{F}' = X_1'+X_2'$ is a 1-approximation of $\mathcal{F}$. Note that both the above results also hold true for 0-approximations, i.e., if $\overline{X_1'}$, $\overline{X_2'}$ are 0-approximations of X_1, X_2 then (i) $\mathcal{F}' = \overline{X_1'}\,\overline{X_2'}$ is a 0-approximation of $\mathcal{F} = X_1X_2$, and (ii) $\mathcal{F}' = \overline{X_1'} + \overline{X_2'}$ is a 0-approximation of $\mathcal{F} = X_1 + X_2$. The above theorems can be generalized to n variables using induction on n. Using this theorem, it can be proved that the exact cube selection algorithm described below always gives rise to the correct approximation at the primary outputs.

Exact cube selection: This technique derives an approximate logic function by picking a subset of cubes from the SOP expression of type 0 and type 1 nodes. Type EX and type DC nodes are left untouched. The idea is to pick cubes to maximize the approximation percentage for low overhead. The phase of the SOP expression used for cube selection must match the node type for correctness of approximation. For instance, if the node type is 0, the SOP expression must be written in the zero phase before selecting the cubes.

A cube is said to conform to a fanin node of type 0(1) if the literal in the cube corresponding to the fanin node is a '0'('1') or '–' (don't care). A cube conforms to a fanin node of type DC if the corresponding literal in the cube is '–'. Every cube conforms to a fanin node of type EX. A cube is selected only if it conforms to the type assignment of every fanin node. It is proved using the theorem above that this method always generates a correct approximate function. This method uses only a subset of the feasible subspace because it does not exploit the observability don't care space to identify cubes in the feasible subspace that do not conform to the fanin node types. This is exploited in the cube selection method based on observability don't cares described below.

Observability don't care based selection: This technique uses local observability don't cares to expand the space from which cubes can be selected. Local observability don't cares refers to the observability don't care space with respect to the output of the node, as opposed to the primary output of the circuit (deriving which is computationally intensive). Equation 1 shows the Boolean expression for computing the feasible subspace. For simplicity, the Boolean expression for a node with two fanin nodes, x_1 of type 1 and x_2 of type 0 is shown.

$$\begin{aligned}
&\mathcal{F}(x_1 + \overline{\mathrm{Obs}_{x_1}})(\overline{x_2} + \overline{\mathrm{Obs}_{x_2}}) && \text{for a node of type 1}\\
&\mathcal{F} + \left(\overline{(x_1 + \overline{\mathrm{Obs}_{x_1}})(\overline{x_2} + \overline{\mathrm{Obs}_{x_2}})}\right) && \text{for a node of type 0}
\end{aligned} \tag{1}$$

$\mathcal{F}$ is the local Boolean function of the node. The term $(x_1+\overline{\mathrm{Obs}_{x_1}})$ picks cubes that either conform to the type of the fanin node x_1 or those in which x_1 is not observable. For a node of type DC, only the observability don't care term is used. As long as only a single input bit flips in the input vector of a node, this technique guarantees correctness. However, multiple bit flips may render the approximation incorrect because the observability of multiple bit flips is computed as the Boolean AND of the observabilities of the bits instead of their joint observability (which is computationally intensive). Hence, this technique may include minterms outside the feasible subspace. Multiple bit flips at the inputs of a node are rare when the Boolean functions at the inputs are well approximated.

Example: The cube selection algorithms are illustrated with an example. Figure 1(a) shows a circuit and its technology-independent representation. The 1-approximation of this network obtained using the exact cube selection algorithm is shown in Fig. 1(b). Two approximate circuits are shown to illustrate the trade-off between approximation percentage and area overhead. In solution 1, nodes n_2 and n_5 were assigned type 1 and the rest of the nodes were assigned type DC. Hence, only one cube was selected from the SOP of n_5 that conformed to the type assignment of its fanin nodes. In solution 2, n_2, n_4 and n_5 were assigned type 1 and the rest of the nodes were assigned type DC. Hence, two cubes were selected from the SOP of n_5 that conformed to the type assignment of its fanin nodes (n_2, n_3, and n_4). For the same type assignment, cube selection based on observability don't cares (Fig. 1(c)) discovered an additional cube -11 in the 1-minterm space of n_5. This minterm is acceptable in the observability don't care based algorithm because the type DC nodes (n_3 and n_4) are not observable in this minterm. Note that cube selection based on observability don't cares gives the solutions shown in Fig. 1(b) in addition to the solution in Fig. 1(c), and thus explores a richer space than the exact cube selection approach. However, the disadvantage of the observability don't-care based cube selection is that it does not guarantee a correct approximation, unlike the exact cube selection approach.

2.2 Iterative cube selection algorithm

Both techniques for cube selection proposed above have their merits and demerits. The exact cube selection algorithm guarantees the correctness of the approximate function. However, this is accomplished at the cost of imposing strict constraints on cube selection, thus affecting the approximation percentage. The observability don't care based cube selection approach relaxes the constraints on cube selection but does not guarantee correctness of the approximate function. Here, we propose an algorithm that uses these techniques iteratively to achieve correctness while maintaining a high approximation percentage. The algorithm takes as input a network in which the type assignment of nodes has been done. The SOPs of type 0 and type 1 nodes are rewritten in the correct form so that the phase of the SOP matches the type of the node. The algorithm is split into two stages:

Approximation of SOPs: The SOP of every node (including type EX and type DC) is reduced by discarding cubes whose contribution to the Boolean function is insignificant. Usually, the cubes with a large support set fall into this category. For instance, in the Boolean function $a+\bar{b}c+b\bar{c}d\bar{e}$, the contribution of the cube $b\bar{c}d\bar{e}$ is least significant and is discarded. This cube selection is done freely without any constraint to ensure correctness. The aim of this stage is to lower area overhead while maintaining a high approximation percentage. The primary outputs of the circuit are then checked for correctness of approximate functions. This can be done very efficiently using SAT algorithms, or by checking the implication condition for correct approximation using BDDs of the original and approximated outputs. If all the outputs have been correctly approximated, then we are done. Otherwise, the outputs that have been incorrectly approximated are corrected in the second stage.

Ensuring correctness: The input to this stage is a primary output of the circuit that has been incorrectly approximated by the previous stage. The first step in correcting the approximation at the output is to perform a backward traversal of the circuit to identify one or more potential sources of incorrect approximation. A node is a source of incorrect approximation if the Boolean function of the node has been incorrectly approximated but all its fanin nodes have been correctly approximated. The first attempt at correcting the approximation at this node is made by selecting the cubes using observability don't cares. If it fails to correct the approximation at the node, the exact cube selection approach is used, which guarantees a correct approximation. The approximation at the primary output is checked again for correctness, and this procedure is repeated for other sources of incorrect approximation until the approximation at the output is fixed.

Note that it is possible that even when the primary outputs of the circuit are correctly approximated, there may be internal nodes that are not approximated correctly. This means that an error in approximation of an internal node is not observable at the primary output of the circuit, which is acceptable. Thus, the algorithm is implicitly able to explore the global observability don't care space of the internal nodes.

The computational complexity of the iterative algorithm depends on the amount of backtracking that needs to be performed in order to ensure correctness of approximation. For the benchmark circuits considered in this paper, the primary outputs were correctly approximated after the first stage of the algorithm in most cases, i.e., there was no need for backtracking to fix the approximations at the nodes. This can be attributed to the type assignment done in the preprocessing stage. Thus, a good type assignment and a high approximation percentage at the internal nodes makes it possible to achieve correctness without any backtracking.

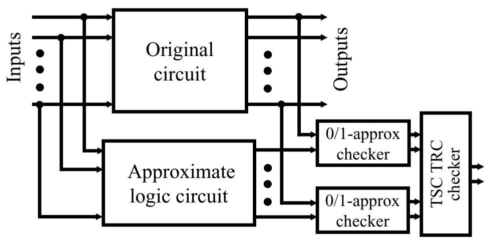


Figure 2: CED based on approximate logic circuits

3. Approximate logic circuits for CED

CED based on approximate logic circuits is illustrated in Fig. 2. Just as in conventional CED, the approximate logic circuit is used as the check symbol generator for the given logic circuit. For every primary output of the circuit, either a 1-approximate or a 0-approximate function is used for detection of $1 \rightarrow 0$ or $0 \rightarrow 1$ errors. The type of approximation for a primary output is decided by the type of error ($1 \rightarrow 0$ or $0 \rightarrow 1$) that dominates at that output and can be determined by reliability analysis. In this paper, the technique for reliability analysis described in [14] was used to compute the $1 \rightarrow 0$ and $0 \rightarrow 1$ errors at the outputs. Since reliability analysis is performed on a technology-mapped netlist, this is done using a quick synthesis and mapping pass on the multi-level technology-independent network. In Sec. 4.1, we show that CED coverage is insensitive to the scripts used for synthesis of the original circuit as well as the specific library used for quick synthesis and mapping prior to reliability analysis. Following reliability analysis, the appropriate 0/1-approximate logic circuit is synthesized using the algorithm described in Sec. 2.2. Each primary output and its corresponding output in the approximate logic circuit is checked using a 0/1-approximate checker, the design and implementation of which is described in Sec. 3.2. The outputs of the 0/1-approximate checkers are consolidated using a conventional totally self-checking (TSC) two-rail code (TRC) checker [3].

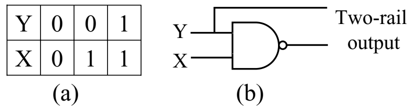
3.1 Low overhead intrusive CED

Although the discussion so far has focused on using approximate logic circuits for non-intrusive CED, further reductions in area-power overhead can be achieved as follows. By merging structurally or functionally equivalent nodes between the original and the approximate logic circuit, it is possible to trade-off CED coverage for further reductions in overhead. This sharing of logic between the original and approximate logic circuits makes CED intrusive. Partial duplication for CED described in [10] can be viewed as a special case of approximate logic functions with logic sharing. In partial duplication, the approximate logic function has a 100%

approximation percentage and non-critical nodes are shared between the original and the approximate logic circuits. Thus, results for CED coverage using partial duplication are a lower bound for the CED coverage achievable using approximate logic circuits with logic sharing and are used as the basis for comparison in Sec. 4.

3.2 Totally self-checking checker

Checker design is an integral part of concurrent error detection. The function of the checker is to monitor the output of the circuit and the check symbol generator, and to signal an error when they do not form a valid codeword. Checkers are usually designed to be totally self-checking (TSC) by satisfying the code-disjoint, fault-secure, and self-testing properties w.r.t a specified fault class. Consider a circuit with an output Y protected by a 0-approximate (because a $0 \rightarrow 1$ error at Y is dominant) logic circuit with output X. Denote the Boolean functions at Y and X by $\mathcal{F}_y$ and $\mathcal{F}_x$. By definition of 0-approximation, $\overline{\mathcal{F}_x} \Rightarrow \overline{\mathcal{F}_y}$ and $\mathcal{F}_y \Rightarrow \mathcal{F}_x$. Thus, when $X\!=\!0$ the approximate logic circuit protects the output Y against $0 \rightarrow 1$ errors, and CED is active. A small fraction of undetected faults arise when X=1, and CED is not active. The proposed checker 0-approximate checker, shown in Fig. 3 (b), is TSC w.r.t all single stuck-at faults when CED is active. When CED is not active, there are exceptions where the checker violates the TSC property.



Y	0	0	1
X	0	1	1

Figure 3: Totally self-checking checker design

Code-disjointness ensures that the checker gives an invalid output codeword when an invalid codeword is presented at its inputs. The valid input codeword space for the checker is shown in Fig. 3(a). Since, X is a 0-approximation of Y, the input codeword space does not contain $X = 0, Y = 1$. The output codeword space is the two-rail code, i.e., $\{01, 10\}$. This is desirable because the outputs of the checkers can be consolidated using a two-rail code checker. It is evident from Fig. 3(a) that the checker is code-disjoint.

Self-testing ensures that all the faults in the specified fault class are testable under normal operation. Since CED with approximate logic circuits protects only unidirectional errors, errors due to faults in the unprotected direction (Y stuck-at 0 and X stuck-at 1 for a 0-approximation) violate the self-testing property. Since $\mathcal{F}_x$ is a 0-approximation of $\mathcal{F}_y$, the fault Y stuck-at 0 will always violate the self-testing property as X=1, Y=1 is the only input vector that can test Y stuck-at 0. However, for Y stuck-at-0, the input to the checker becomes X=1, Y=0, which is a valid codeword and so cannot detect the fault. Hence, no input vector under normal operation can detect Y stuck-at 0. Similarly, since $\mathcal{F}_y$ is a 1-approximation of $\mathcal{F}_x$ if $\mathcal{F}_x$ is a 0-approximation of $\mathcal{F}_y$, the fault X stuck-at-1 will always violate the self-testing property. Note that since the checker is an irredundant logic circuit, it is completely testable offline for all single stuck-at faults through incorporation into a scan chain.

Fault-secureness ensures that a fault within the fault class either gives the fault-free response or an invalid codeword at the output of the checker. The checker maps an asymmetric input codeword space to the dual-rail codespace at the output. Since X is a 0-approximation of Y, the checker is not fault secure for stuck-at faults at Y when X=1.

4. Results

In this section, we present simulation results for CED using approximate logic circuits. The framework for synthesizing approximate logic circuits was implemented in ABC, the logic synthesis tool developed at Berkeley [15]. The simulations were run on a 64-bit 2.4 GHz Opteron-based system with 6 GB memory. The results for error detection in MCNC benchmark circuits are presented in this section. The inputs to the circuits are assumed to have an equal probability of occurrence, i.e., there is no input vector biasing.

The results in Table 1 and Table 2 are reported for the single stuck-at fault model with all the gates in the circuit having the same probability of failure. Using this fault model, the circuit is simulated with a randomly injected single stuck-at fault and a random input vector, for 6.4M runs. CED coverage is the percentage of runs for which an error at the output is detected by the CED technique.

Table 1 shows results for single output cones extracted from MCNC benchmarks. The first column is the name of the benchmark circuit from which the output cone was extracted. The second column is the number of gates in the output cone. The third column is the area overhead of the synthesized approximate logic circuit. The fourth column is the approximation percentage achieved by the approximate logic function. The fifth column is the maximum error detection percentage that can be achieved by protecting the dominant error ($0 \rightarrow 1$ or $1 \rightarrow 0$) at the output. The sixth column reports the error detection percentage, i.e., the CED coverage achieved by the synthesized approximate logic circuit. The results illustrate the effectiveness of the approximate logic functions in achieving a high approximation percentage for low area overhead. The disparity in the approximation percentage and CED coverage for circuits `des` and `i8` is because CED coverage is limited by the amount of skew in the type of errors ($0 \rightarrow 1$ vs. $1 \rightarrow 0$) at the output.

Table 1: Approximation percentage and CED coverage for output cones extracted from benchmark circuits.

Name	Gates	Area (%)	Approx. (%)	CED coverage (%)	
				Max.	Achieved
i8	106	28	80	65	50
des	191	2.7	95.6	56	48
dalu	862	25	93.8	85	71
i10	1141	1.5	91	76	64

The results for error detection in complete MCNC benchmark circuits are shown in Table 2. The proposed synthesis algorithms for approximate logic functions were evaluated on logic benchmarks from this suite. Logic benchmarks with a reasonably large skew in the errors at the outputs have been chosen. Three metrics — area, power overhead and CED coverage have been chosen for comparing the proposed CED technique with existing CED approaches. Area is computed as the total number of gates in the circuit, power is computed as the total switching activity of the gates in the circuit, and CED coverage is computed using fault injection and simulation as described above. Area, power overhead and CED coverage for completely non-intrusive approximate logic functions are shown under the column "No logic sharing". The trade-off between area overhead and CED coverage achieved by merging of non-critical nodes between the original and approximate logic circuit is shown under the column "With logic sharing". These are compared to two existing approaches — intrusive CED based on partial duplication [10] and non-intrusive CED based on single-bit parity checkers. The results show that the same CED coverage can be achieved with the proposed technique with an area overhead that is lower than that for partial duplication. Furthermore, the proposed technique is completely non-intrusive and incurs zero performance penalty because the approximate logic function always has a lower delay on the critical path. For the benchmarks studied in this paper, the delay of the approximate logic circuit was 38% less than the delay of the original circuit on average. The proposed technique is

Table 2: **Area-power overhead and CED coverage for MCNC benchmark circuits.**

Name	Gates	Max. CED coverage	No logic sharing			With logic sharing		Partial duplication [10]			Parity prediction		
			Area	Power	CED coverage	Area	CED coverage	Area	Power	CED coverage	Area	Power	CED coverage
cmb	57	99.7	32	26	98	29	98	48	32	98	87	43	66
cordic	116	88	28	37	82	24	82	26	22	82	29	33	71
term1	260	82	15	25	71	13	70	17	19	70	100	101	92
x1	442	78	36	45	68	26	65	30	37	68	125	120	86
i2	440	89	5	6	84	3	83	6	4	82	100	100	100
frg2	1089	90	30	47	80	22	75	46	48	79	161	133	91
dalu	1166	92	21	35	80	15	77	44	44	77	110	109	94
i10	2866	85	36	56	81	30	77	54	49	81	139	135	64

also scalable and the runtime for the synthesis of the approximate logic circuit for the largest benchmark circuit, `i10`, was 5m28s. Single-bit parity prediction requires average area and power overhead of 106% and 97%, which is roughly 3X higher than the proposed solution, for a 2% improvement in CED coverage on average. In benchmark circuits `cmb` and `i10`, single-bit parity prediction has higher area and power overheads but lower CED coverage as compared to both approximate logic functions and partial duplication. In addition, single-bit parity prediction produces circuits with higher critical path delay. For the benchmark circuits used in this paper, the critical path delay of a single-bit parity prediction circuit was 51% higher than the original circuit on average.

4.1 Technology-independence

In this section, we present results in Table 3 to show that the CED coverage of the proposed technique is not significantly affected by the (i) synthesis scripts used to optimize and map the original and approximate logic circuits or (ii) the library used to map and perform reliability analysis on the original circuit. Initially, reliability analysis was performed on a netlist obtained by quick synthesis to determine the type of approximation for each output. The results of reliability analysis were used to perform synthesis of the approximate logic circuits. Five different technology-mapped implementations of the original circuit were generated using different optimization scripts in ABC and different technology libraries. The same approximate logic function (mapped with the technology library of the original circuit) was used to provide CED for each of the implementations. Table 3 shows the CED coverage for different technology-mapped implementations of the original and approximate logic circuits for the same area-power overheads. The table illustrates that the CED coverage remains fairly constant for different technology-mapped implementations. Thus, we can conclude that the effectiveness of CED achieved using the proposed technique depends mainly on the Boolean function being approximated, i.e., it is technology-independent.

Table 3: Technology-independence of CED coverage

Name	CED coverage %				
	Impln 1	Impln 2	Impln 3	Impln 4	Impln 5
cmb	95.8	96	96.6	95.1	96.7
cordic	74	74.5	74.1	74.6	73
term1	70	73	75	80	71
x1	67.8	68.6	64.1	64.5	68
i2	79	84	82	85	83
frg2	70	69	71.3	76.1	75.2
dalu	71.2	72.1	73	72.4	75
i10	70	71.2	70.5	71.7	72.2

5. Conclusions

As reliability emerges as a first-order constraint in scaled CMOS technologies, there is significant interest in the development of cost-effective techniques for robust error-resilient design of logic circuits. This paper described CED based on the synthesis of approximate logic circuits as a low overhead, non-intrusive solution to enhance reliability. Areas for future research include synthesis of approximate logic circuits for (i) CED of errors caused by delay faults on speed-paths in logic circuits and (ii) combined error detection and error masking to enhance circuit reliability.

References

[1] J. D. Meindl *et al.*, "Limits on silicon nanoelectronis for terascale integration," *Science*, vol. 293, pp. 2044–2049, Sep. 2001.

[2] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, pp. 10–16, 2005.

[3] D. K. Pradhan, ed., *Fault-tolerant computing: Theory and techniques*, vol. 1. Prentice-Hall, NJ, USA, 1986.

[4] S. Kundu and S. M. Reddy, "On symmetric error correcting and all unidirectional error detecting codes," *IEEE Trans. Computers*, vol. 39, pp. 752–761, Jun. 1990.

[5] M. Gössel and S. Graf, *Error detection circuits*. McGraw-Hill Book Company, London, UK, 1993.

[6] N. K. Jha and S. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE Trans. Computer-aided Design*, vol. 2, pp. 878–887, 1993.

[7] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Trans. Computer-aided Design*, vol. 16, pp. 783–789, Jul. 1997.

[8] V. V. Saposhnikov *et al.*, "Self-dual duplication for error detection," in *Proc. Asian Test Symposium*, pp. 296–300, 1998.

[9] D. Das and N. Touba, "Synthesis of circuits with low cost concurrent error detection based on Bose-Lin codes," in *Journal of Electronic Testing: Theory and Applications*, vol. 15, pp. 145–155, 1999.

[10] K. Mohanram and N. A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *Proc. Intl. Test Conference*, pp. 893–901, 2003.

[11] S. Almukhaizim, P. Drineas, and Y. Makris, "Entropy-driven parity tree selection for low-cost concurrent error detection," *IEEE Trans. Computer-aided Design*, vol. 25, pp. 1547–1554, 2006.

[12] G. Hachtel and F. Somenzi, *Logic synthesis and verification*. Kluwer Academic Publishers, 2000.

[13] W. Kunz and D. Pradhan, "Recursive learning: A new implication technique for efficient solutions to CAD problems — Test, verification, and optimization," in *IEEE Trans. Computer-aided Design*, vol. 13, pp. 1143–1158, 1994.

[14] M. Choudhury and K. Mohanram, "Accurate and scalable reliability analysis of logic circuits," in *Proc. Design Automation and Test in Europe*, pp. 1454–1459, 2007.

[15] "ABC Logic synthesis tool." Please visit the URL http://www.eecs.berkeley.edu/∼alanmi/abc/ for further details.