TAMPERE UNIVERSITY OF TECHNOLOGY

HELENA ASTOLA

COMBINING ERROR-CORRECTING CODES AND DECISION DIAGRAMS FOR THE DESIGN OF FAULT-TOLERANT LOGIC

Master of Science Thesis

Examiners: Prof. Ioan Tabus
Prof. Radomir Stanković
Subject approved in the meeting of
The Faculty Council of Computing and
Electrical Engineering on April 6, 2011

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HELENA ASTOLA : Combining Error-Correcting Codes and Decision Diagrams for the Design of Fault-Tolerant Logic
Master of Science Thesis, 63 pages
September 2011
Major: Signal Processing
Examiners: Prof. Ioan Tabus and Prof. Radomir Stanković
Keywords: Error-correcting codes, logic design, fault-tolerant circuits, decision diagrams

In modern logic circuits, fault-tolerance is increasingly important, since even atomic-scale imperfections can result in circuit failures as the size of the components is shrinking. Therefore, in addition to existing techniques for providing fault-tolerance to logic circuits, it is important to develop new techniques for detecting and correcting possible errors resulting from faults in the circuitry.

Error-correcting codes are typically used in data transmission for error detection and correction. Their theory is far developed, and linear codes, in particular, have many useful properties and fast decoding algorithms. The existing fault-tolerance techniques utilizing error-correcting codes require less redundancy than other error detection and correction schemes, and such techniques are usually implemented using special decoding circuits.

Decision diagrams are an efficient graphical representation for logic functions, which, depending on the technology, directly determine the complexity and layout of the circuit. Therefore, they are easy to implement.

In this thesis, error-correcting codes are combined with decision diagrams to obtain a new method for providing fault-tolerance in logic circuits. The resulting method of designing fault-tolerant logic, namely error-correcting decision diagrams, introduces redundancy already to the representations of logic functions, and as a consequence no additional checker circuits are needed in the circuit layouts obtained with the new method. The purpose of the thesis is to introduce this original concept and provide fault-tolerance analysis for the obtained decision diagrams.

The fault-tolerance analysis of error-correcting decision diagrams carried out in this thesis shows that the obtained robust diagrams have a significantly reduced probability for an incorrect output in comparison with non-redundant diagrams. However, such useful properties are not obtained without a cost, since adding redundancy also adds complexity, and consequently better error-correcting properties result in increased complexity in the circuit layout.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HELENA ASTOLA: Combining Error-Correcting Codes and Decision Diagrams for the Design of Fault-Tolerant Logic
Diplomityö, 63 sivua
Syyskuu 2011
Pääaine: Signaalinkäsittely
Tarkastajat: Prof. Ioan Tabus ja Prof. Radomir Stanković
Avainsanat: Virheitäkorjaavat koodit, logiikkasuunnittelu, vikasietoiset piirit, päätösdiagrammit

Moderneissa logiikkapiireissä vikasietoisuuden merkitys on entistä suurempi, sillä komponenttien pienentyessä jopa atomitason epätäydellisyydet voivat aiheuttaa häiriöitä piirien toiminnassa. Olemassaolevien tekniikoiden lisäksi onkin tärkeää kehittää uusia menetelmiä, joilla häiriöistä johtuvat virheet voidaan havaita ja korjata.

Virheitäkorjaavia koodeja käytetään tyypillisesti tiedonsiirrossa aiheutuvien virheiden havaitsemiseen ja korjaamiseen. Näiden koodien teoria on pitkälle kehittynyttä, ja erityisesti lineaarisilla koodeilla on useita hyödyllisiä ominaisuuksia ja tehokkaita dekoodausalgoritmeja. Logiikkapiirien vikasietoisuuden lisäämisessä käytetyt menetelmät, jotka pohjautuvat virheitäkorjaaviin koodeihin, toteutetaan tavallisesti liittämällä piiristöön erityisiä dekoodauksen suorittavia piirejä, ja nämä menetelmät vaativat usein vähemmän redundanssia kuin muut vastaavat tekniikat.

Päätösdiagrammit ovat tehokas keino esittää logiikkafunktioita, jotka toteutuksen teknologiasta riippuen määrittävät suoraan suunniteltavan piirin kompleksisuuden, ja niiden toteuttaminen piiritasolla on helppoa.

Tässä opinnäytetyössä virheitäkorjaavat koodit ja päätösdiagrammit yhdistämällä on kehitetty uudenlainen tekniikka vikasietoisen logiikan suunnitteluun. Kehitetyllä menetelmällä redundanssi saadaan jo logiikkafunktioiden esitysmuotoihin eikä valmiissa piireissä näin ollen tarvita ylimääräisiä virheiden havainnoinnin tai korjaamisen suorittavia osia. Opinnäytetyön tarkoituksena on esittää tämä uusi menetelmä ja arvioida saatujen päätösdiagrammien vikasietoisuutta.

Opinnäytetyössä on mallinnettu kehitetyn menetelmän suorituskykyä vikasietoisuuden kannalta ja osoitettu, että suunnittelun tuloksena saaduilla virheitäkorjaavilla päätösdiagrammeilla on huomattavasti pienempi todennäköisyys virheellisiin ulostuloihin kuin vastaavilla tavanomaisilla diagrammeilla. Redundanssin lisääminen kuitenkin lisää myös kompleksisuutta, ja tehokkaamman virheenkorjauksen myötä piirin kompleksisuus kasvaa. Kuitenkin jo maltillisella kompleksisuuden lisäämisellä saavutetaan huomattavia parannuksia piirin vikasietoisuudessa.

PREFACE

This Master of Science thesis has been written for the Department of Signal Processing at the Tampere University of Technology. The research done for this thesis in the Department of Signal Processing has also resulted in two research papers, [4] and [5], which introduce and analyze the fault-tolerant logic design application discussed in this thesis.

I wish to thank my supervisors Prof. Ioan Tabus and Prof. Radomir Stanković for their valuable comments and guidance, and Ph.D. Stanislav Stanković for his help and taking part in the research. I am also grateful to Prof. Jaakko Astola for his help and support.

Tampere, August 15, 2011

Helena Astola
helena.astola@tut.fi

CONTENTS

ABBREVIATIONS AND NOTATION

$\oplus$	Logical exclusive OR.
$\mathbf{c}$	A codeword.
$d_H(\mathbf{x}, \mathbf{y})$	The Hamming distance between the vectors $\mathbf{x}$ and $\mathbf{y}$.
$d_L(\mathbf{x}, \mathbf{y})$	The Lee distance between the vectors $\mathbf{x}$ and $\mathbf{y}$.
e	The number of errors a code corrects.
$\mathbb{F}_q$	The finite field of q elements.
$\mathbb{F}_q^i$	The vector space of length i vectors over $\mathbb{F}_q$.
$\mathbf{G}$	A generator matrix of a linear code.
$\mathbf{H}$	A parity check matrix of a linear code.
$\mathbf{I}_i$	The identity matrix of size $i \times i$.
k	The number of inputs of a function; the dimension of a linear code, which equals the number of inputs of a non-robust function.
K	The number of repeated memory elements.
l	The number of outputs of a function.
L	The number of subsets of $\{1, 2, \ldots, M\}$.
μ_i	A random variable.
M	The number of non-terminal nodes in a decision diagram.
n	The length of a code, which for robust diagrams equals the number of variables.
N	The number of repeated modules (processors in the (N, K) concept).
p	The probability that a decision node is faulty.
P_i	A subset of the set $\{1, 2, \ldots, M\}$.
q	The size of the domain and range of a q-ary function, i.e., the number of elements in the finite field $\mathbb{F}_q$.
ρ	A binary relation.
r	The number of repetitions in a repetition code.
$\mathbf{s}$	The syndrome of a word $\mathbf{v} \in \mathbb{F}_q^n$.
τ	The covering radius of a code.
w_H	The Hamming weight of a vector.
w_L	The Lee weight of a vector.
BDD	Binary decision diagram.
BDT	Binary decision tree.
DNF	Disjunctive normal form.
ECC	Error-correcting code.
LDPC	Low-density parity-check (code).
MTBDD	Multi-terminal binary decision diagram.

MTDD Multi-terminal decision diagram.
NMR N-modular redundancy.
OBDD Ordered binary decision diagram.
PLA Programmable logic array.
TMR Triple modular redundancy.
TSC Totally self-checking (circuit).

1. INTRODUCTION

The role of digital systems in modern life is increasingly important, and often these systems handle critical information and their accurate performance is essential for a given application. This high dependability is required, for example, in military and aerospace computing. In addition to these areas where the demands are extremely high, in most applications high dependability makes the products more competitive as their digital circuits perform their designed functions with a lower error rate. In modern logic circuits, the transistors are shrinking, which means that even atomic-scale imperfections in each transistor can have a negative effect on the performance of these circuits. Therefore, in addition to existing techniques against failures in digital systems, it is important to develop new techniques for detecting and, in particular, correcting errors resulting from different types of faults in digital circuits.

Due to the importance of fault-tolerance in digital systems, numerous techniques have been developed against hardware failures. The most well-known such technique is triple modular redundancy, for which the groundwork was laid by von Neumann in [35]. It is a technique, which introduces redundancy to logic circuits by module triplication. Several authors have later further analyzed and developed this technique (see, for instance, [2], [17]).

Error-correcting codes have many useful properties and they are most typically used in data transmission to detect and correct errors on noisy communication channels [34]. With error-correcting codes, error detection and correction in logic circuits can often be implemented with less redundancy than when using other methods [26]. The properties of error-correcting codes are widely studied, and the highly developed theory behind them makes them a fruitful basis for new error detection and correction schemes. Coding theory has been exploited in several techniques for providing fault-tolerance in logic circuits, usually by introducing special circuits into logic modules, which handle the detection and correction of possible errors [26].

Since error correction in logic circuits is increasingly important, it is essential to find systematic ways to increase fault-tolerance already in the representations of switching functions, i.e., functions realized by the circuits. The method for providing fault-tolerance introduced in this thesis combines the theory of error-correcting codes and decision diagrams to obtain robust representations for functions, which are easily implemented with the suitable technology. The main advantage of using decision

diagrams for representing switching functions is that the layout and complexity of a circuit is directly determined by the decision diagram. The idea is to create error-correcting decision diagrams, i.e., a way of representing switching functions in a robust manner that can directly be mapped to technology. The purpose of the thesis is to introduce this original concept and provide fault-tolerance analysis for the obtained decision diagrams. The idea of combining error-correcting codes and decision diagrams introduced in this thesis has also resulted in two conference papers [4], [5], where error-correcting decision diagrams and their performance is discussed.

The thesis is structured as follows. In Chapter 2, the mathematical background for the topics discussed in this thesis is reviewed. This includes the definitions for decision diagrams and the theory of error-correcting codes, as well as some basic concepts related to discrete functions and their representations. Fault-tolerance in digital systems is explained in more detail in Chapter 3, where also existing error correction and detection schemes, in particular those utilizing coding theory, are discussed.

In Chapter 4, an original technique, namely error-correcting decision diagrams, for providing fault-tolerance in logic circuits is presented for binary and multiple-valued logic. The performance of this technique is analyzed in Chapter 5, where the probabilities for correct outputs in such constructions are determined. Since determining exact probabilities for larger configurations is very time-consuming, a method of approximating these probabilities is also introduced. Further discussion based on the analysis is included in Chapter 6.

2. MATHEMATICAL BACKGROUND

In this chapter, the mathematical background for the topics in this thesis is provided. Since digital circuits realize discrete functions, we begin by defining discrete functions. Then, decision diagrams are introduced as representations of discrete functions. The fault-tolerant circuit designs introduced in this thesis are derived using the theory of error-correcting codes, which are explained in the final section of this chapter. However, before moving on to the theory of error-correcting codes, some basic definitions regarding fields and vector spaces are provided for a better understanding of the theory.

2.1 Discrete Functions

The functions discussed in this thesis belong to the class of discrete functions, which can be defined in the following way [6]:

Definition 2.1 *Let A and B be sets. Let ρ be a binary relation from A to B. If for every element $a \in A$ there exists a unique element in $b \in B$ such that $(a, b) \in \rho$, then ρ is a function from A to B, which is denoted by $\rho : A \to B$. If A and B are finite, then ρ is a discrete function.*

The applications in this thesis concern discrete functions on finite sets, i.e., functions for which the domain A and range B are finite. A function f of k variables (inputs) is denoted as $f(x_0, x_1, \ldots, x_{k-1})$, $x_i \in A$, $f(x_0, x_1, \ldots, x_{k-1}) \in B$, and a multi-output function is defined as:

Definition 2.2 *Let A and B be finite sets, $k \geq 0$, $l \geq 0$ and $f : A^k \to B^l$ a function. Then f is a multi-output function.*

Thus, a multi-output function is equivalent to a system of single-output functions $f = (f_0, f_1, \ldots, f_{l-1})$.

The most commonly used functions in digital logic are switching (Boolean) functions, i.e., functions $f : \{0,1\}^k \to \{0,1\}$, which describe the behavior of binary logic circuits. Logic systems with multiple inputs or outputs are represented by multi-output switching functions that are functions of the form $f : \{0,1\}^k \to \{0,1\}^l$.

Multiple-valued functions are functions with a domain A^k and range B^l, where $|A| = |B| = q > 2$. For example, ternary functions are a class of functions of the

form $f : \{0,1,2\}^k \rightarrow \{0,1,2\}^l$. Hence, for ternary functions, $q = 3$. Functions with $q = 4$ are called quaternary functions. Generally, a function with a domain having q values is a q-ary function. Notice that in this thesis we only refer to q-ary functions, where q is the number of elements of the finite field $\mathbb{F}_q$ (see, Section 2.3), but in general, q can be any integer, which is larger than 2.

There exist several methods of representing discrete functions, e.g. truth-tables, algebraic expressions, and graphic representations. Decision diagrams are a graphic method of representing discrete functions, and they are explained in detail in Section 2.2. For understanding decision diagrams, certain representations and expressions of logic functions are explained here.

Consider a switching function $f(x_0, x_1, x_2) : \{0,1\}^3 \rightarrow \{0,1\}$. This function can be given by listing its values as (x_0, x_1, x_2) run through the values of the domain $\{0,1\}^3$. For a binary switching function, listing these values into a tabular form is called a truth table (Table 2.1). The function $f_1 = f(x_0, x_1, x_2)$ may also be represented by a vector of the function values, which in case of switching functions is called a truth-vector. The truth-vector of the switching function f_1 defined in Table 2.1 is $\mathbf{F}_1 = [0,1,1,1,0,1,0,1]^T$. When representing switching functions with truth-vectors, the ordering of the 2^k binary input sequences should be specified. Unless otherwise stated, we use lexicographic ordering as in Table 2.1.

Table 2.1: The truth-table of a binary switching function f_1.

$x_0x_1x_2$	$f_1(x_0, x_1, x_2)$
000	0
001	1
010	1
011	1
100	0
101	1
110	0
111	1

Switching functions can also be represented as a formula written in terms of some operations over an algebraic structure. For these representations, some basic definitions must be introduced. The following definitions are given for switching functions but can directly be generalized to q-ary functions [6].

A two-valued variable x_i may be written in terms of a positive literal x_i or a negative literal $\bar{x}_i$. A positive literal is just an atom, which is a logical formula containing no subformulas, and a negative literal is the negation of an atom [29]. Denote by $\cdot$ the logical AND operation corresponding to a product of variables and by $+$ the logical OR operation corresponding to a sum of variables. Any switching

function can be written with literals and operations $\cdot$ and $+$.

For example, we can represent the function f_1 in Table 2.1 as a canonical sum of products, i.e., in the complete disjunctive normal form (DNF), which corresponds to the lines on the table where f has the value 1 ($\cdot$ omitted):

$$f_1 = \bar{x}_0\bar{x}_1x_2 + \bar{x}_0x_1\bar{x}_2 + \bar{x}_0x_1x_2 + x_0\bar{x}_1x_2 + x_0x_1x_2, \tag{2.1}$$

or equivalently in a more compact DNF as

$$f_1 = x_2 + \bar{x}_0x_1. \tag{2.2}$$

The expression in (2.2) is not canonical and can be derived from the sum of the two functions $f_2 = x_2$ and $f_3 = \bar{x}_0x_1$ as shown in Table 2.2.

Table 2.2: The representation of f_1 as the sum of f_2 and f_3.

$x_0x_1x_2$	f_2
000	0
001	1
010	0
011	1
100	0
101	1
110	0
111	1

$+$

$x_0x_1x_2$	f_3
000	0
001	0
010	1
011	1
100	0
101	0
110	0
111	0

$=$

$x_0x_1x_2$	f_1
000	0
001	1
010	1
011	1
100	0
101	1
110	0
111	1

2.2 Decision Diagrams

Decision diagrams are an effective way of representing discrete functions graphically. The idea of representing switching circuits using reduced binary decision diagrams (BDDs) was formalized by Bryant in [9], and the topic has been further explored by numerous authors. Binary decision diagrams have many applications in logic design, e.g., in logic circuit minimization [3] and probabilistic analysis of digital circuits [33]. Decision diagrams can also be used for representing other discrete functions than binary switching functions, namely multiple-valued functions. An important feature of decision diagrams is that they are easily mapped to technology. For example, depending on the technology, the number of gates in the circuit directly relates to the number of nodes in the decision diagram and the delay of the circuit is related to path lengths. In Figure 2.1 is an example of the correspondence of a binary decision diagram to a circuit layout, where the circuit is constructed using multiplexers. More on circuit realization can be found in, for example, [28].

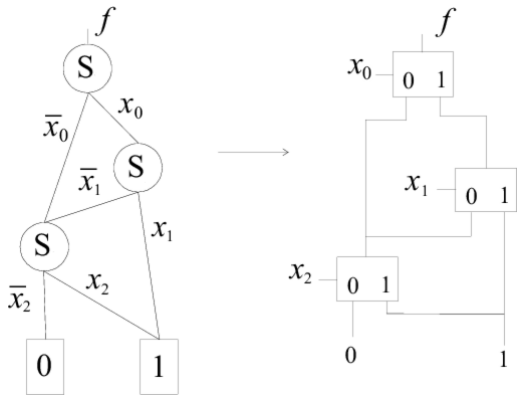


Figure 2.1: Correspondence between BDDs and networks of multiplexers [6].

In this section, the definitions of binary decision diagrams, multiple-valued, i.e., q-ary decision diagrams, multi-terminal decision diagrams (MTDDs) and shared decision diagrams are given. The definitions and basic concepts and properties related to decision diagrams are given according to [6], [23].

2.2.1 Binary Decision Diagrams

Binary decision diagrams are used to represent switching functions, i.e., functions of the form $f : \{0,1\}^k \rightarrow \{0,1\}$. We define binary decision diagrams using binary decision trees, which are graphic representations of functions in the complete DNF.

Definition 2.3 *A binary decision tree (BDT) is a rooted directed graph having $k+1$ levels with two different types of vertices. On level i, where $i = 0, \ldots, k-1$, are the non-terminal nodes, each having two outgoing edges labeled by 0 and 1 or by corresponding literals $\bar{x}_i$ and x_i. On level k are the terminal nodes having the label 0 or 1 and no outgoing edges.*

A BDT has a direct correspondence to the truth-table of a function. Let $f(x_0, x_1, \ldots, x_{k-1})$ be a switching function. In the binary decision tree of f, each node on level i corresponds to a specific variable x_i, and by following the edges the value of the function at $(x_0, x_1, \ldots, x_{k-1})$ is found in the terminal node. Figure 2.2 shows a BDT representing the function f_1 defined in Table 2.1.

Definition 2.4 *A binary decision diagram is a rooted directed graph obtained from a binary decision tree by the following reduction rules:*

1. *If two sub-graphs represent the same function, delete one, and connect the edge pointing to its root to the remaining subgraph.*

2. *If both edges of a node point to the same sub-graph, delete that node, and directly connect its incoming edge to the sub-graph.*

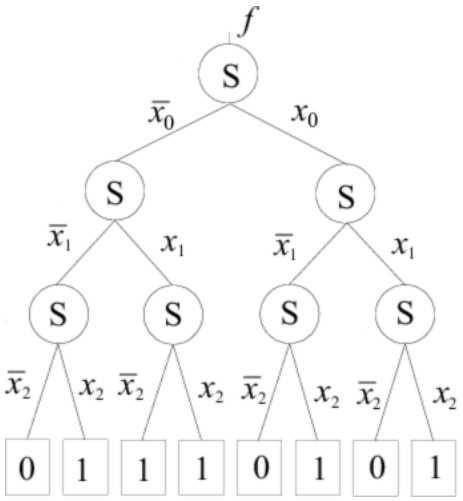


Figure 2.2: A BDT for the function f_1 in Table 2.1.

In Figure 2.3 is a BDD representing the function f_1 defined in Table 2.1.

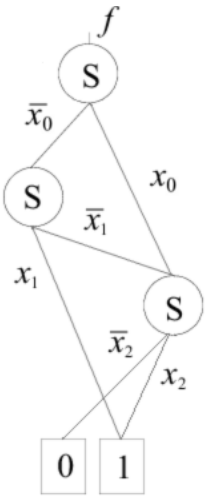


Figure 2.3: A BDD for the function f_1 in Table 2.1.

The letter **S** in the nodes means that the nodes in the diagrams are Shannon nodes, i.e., the decision diagram is a graphic representation of the Shannon expansion of the function, which is defined as follows.

Definition 2.5 *The Shannon expansion of the switching function* $f(x_0, x_1, \ldots, x_{k-1})$ *with respect to the variable* x_i *is* $f = \overline{x}_i f_0 \oplus x_i f_1$*, where* $f_0 = f(x_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{k-1})$ *and* $f_1 = f(x_0, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{k-1})$*, and* $\oplus$ *denotes the logical Exclusive OR.*

As in most literature, when discussing BDDs, we refer to ordered binary decision diagrams (OBDDs), where the variable x_i corresponds to the level i of the decision

tree. In [9], it has been shown that the OBDD of a given function is canonical, i.e., for a given ordering, the OBDD of a given function is unique up to function graph isomorphism, for which the definition was given by Bryant in [9]. Several important consequences follow from the uniqueness of OBDDs, for example, equivalence of functions can easily be tested using OBDDs. The property of the representation of functions as BDDs being canonical means, that when we reduce the BDT of a function with a given variable ordering into a BDD, the operations of reduction cannot be made in such a way or order, that two non-isomorphic BDDs could be obtained starting from the same BDT.

2.2.2 Multiple-Valued Decision Diagrams

The definition of a BDD is easily extended to the q-ary case for representing functions with a larger than two-valued domain. Again, we define the decision diagram using the definition of a decision tree.

Definition 2.6 *A q-ary decision tree is a rooted directed graph having $k+1$ levels with two different types of vertices. On level i, where $i = 0, \ldots, k-1$ are the non-terminal nodes, each having q outgoing edges with labels from the set $\{0, 1, \ldots, q-1\}$ or the set of q-ary literals $X_i^0, X_i^1, \ldots, X_i^{q-1}$. On level k are the terminal nodes, which have labels from the set $\{0, 1, \ldots, q-1\}$ and no outgoing edges.*

Definition 2.7 *A q-ary decision diagram is a rooted directed graph obtained from a q-ary decision tree by the reduction rules in Definition 2.4.*

In Figure 2.4 is an example of the structure of nodes in a q-ary decision diagram when $q = 4$.

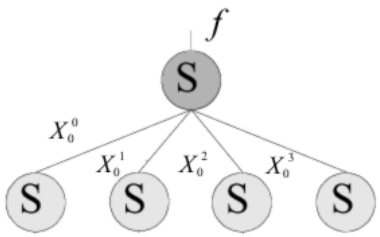


Figure 2.4: The node structure in a quaternary decision diagram.

In the case of a q-ary function $f(x_0, x_1, \ldots, x_{k-1})$, the Shannon expansion of f with respect to the variable x_i is $f(x_0, x_1, \ldots, x_{k-1}) = X_i^0 f(x_0, x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{k-1}) + X_i^1 f(x_0, x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{k-1}) + \cdots + X_i^{q-1} f(x_0, x_1, \ldots, x_{i-1}, q-1, x_{i+1}, \ldots, x_{k-1})$ [30].

2.2.3 Multi-terminal and Shared Decision Diagrams

The number of terminal nodes in decision diagrams is not necessarily limited to q nodes. Such decision diagrams are called multi-terminal decision diagrams and are used to represent functions with a range having more than q elements. The only difference between a decision diagram and a MTDD is the number of terminal nodes. For example, a multi-terminal binary decision diagram (MTBDD) is a BDD having more than two terminal nodes. In other words, a MTBDD represents a function $f : \{0,1\}^k \to B$, where B has more than two elements. The construction of a MTDD or MTBDD can be derived from decision trees similarly as described for BDDs and q-ary decision diagrams.

MTDDs and shared decision diagrams are useful when dealing with multi-output functions or systems of functions, where terminal nodes are labeled by the values that the system can get. For example, for switching functions, the binary l-tuples of the outputs are interpreted as binary representations of the corresponding integers and terminal values are labeled by these integer values. Shared decision diagrams are constructed by first constructing the decision diagrams for the different outputs, and then sharing the isomorphic subgraphs of the obtained decision diagrams.

In Figures 2.5a and 2.5b are examples of a MTBDD and a shared BDD. Both diagrams represent a two-output function $f = (f_0, f_1)$ where $f_0 = x_0$ and $f_1 = \bar{x}_0 x_1$. In the MTBDD, the output values are interpreted as the corresponding integers. For example, with the input $(0,1)$, the output values are $f_0 = 0$ and $f_1 = 1$, which in the MTBBD is interpreted as the binary representation 01, which corresponds to the integer value 1.

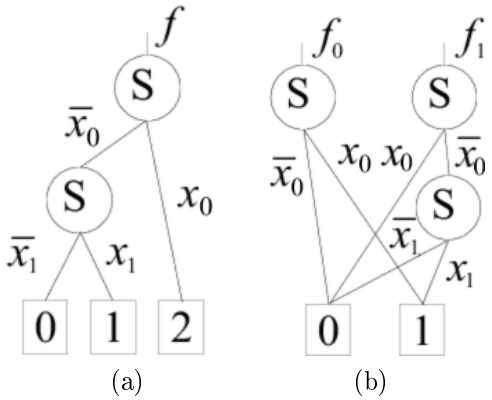


Figure 2.5: A MTBDD (a) and a shared binary decision diagram (b) [6].

2.3 Fields and Vector Spaces

Linear error-correcting codes are defined as subspaces of vector spaces over fields. Therefore, some basic concepts related to fields and vector spaces must be provided

here. The following definitions and properties are given according to [13], [20].

A field is defined using the definition of a group, which can be stated as:

Definition 2.8 *A group $(G, *)$ is a set G together with a binary operation $*$ on G, such that the following properties for G and $*$ are satisfied:*

1. *The set G is closed under the operation $*$, i.e., for $a, b \in G$ the result of the operation $a * b$ is also in G.*
2. *The operation $*$ is associative, i.e., $(a * b) * c = a * (b * c)$.*
3. *There is an element $e \in G$ such that $e * a = a * e = a$ for all $a \in G$. The element e is called the identity element of G.*
4. *For each $a \in G$ there is an element $a' \in G$ such that $a * a' = a' * a = e$. The element a' is called the inverse of a.*

The group G is called Abelian if commutativity, i.e., $a*b = b*a$ holds for all $a, b \in G$. The identity element (neutral element) of an Abelian group is usually denoted by 0. A subset H of the set G is a subgroup if it also forms a group under the operation $*$.

Definition 2.9 *If $(G, *)$ is a group, g an element of G, and H a subgroup of G, then $gH = \{g * h : h \in H\}$ is a left coset of H in G and $Hg = \{h * g : h \in H\}$ is a right coset of H in G.*

Cosets appear in the decoding process of linear codes. For Abelian groups, the left and right cosets coincide.

Definition 2.10 *A field is a set $\mathbb{F}$ together with two operations, "$+$" and "$\cdot$", such that the set is an Abelian group under the operation "$+$", the nonzero elements of the set form an Abelian group under the operation "$\cdot$" and the distributive law $a \cdot (b + c) = a \cdot b + a \cdot c$ holds.*

The multiplicative identity element (the identity element of the Abelian group under "$\cdot$") of a field is called the unit element and it is usually denoted by 1.

Definition 2.11 *Let $\mathbb{F}$ be a field. A vector space V over the field $\mathbb{F}$ is an additive Abelian group, together with the association*

$$(x, v) \to xv,$$

where $x \in \mathbb{F}$ and $v \in V$, satisfying:

1. *If 1 is the unit element of $\mathbb{F}$, then $1v = v$ for all $v \in V$.*

2. *If* $c \in \mathbb{F}$ *and* $v, w \in V$*, then* $c(v + w) = cv + cw$.

3. *If* $x, y \in \mathbb{F}$ *and* $v \in V$*, then* $(x + y)v = xv + yv$.

4. *If* $x, y \in \mathbb{F}$ *and* $v \in V$*, then* $(xy)v = x(yv)$.

Definition 2.12 *A subset* W *of* V *is a subspace of* V *if the following conditions are satisfied:*

1. *If* $v, w \in W$*, their sum* $v + w$ *is also an element of* W.

2. *The identity element* 0 *of* V *is an element of* W.

3. *If* $v \in W$ *and* $c \in \mathbb{F}$*, then* $cv \in W$.

We denote a field of q elements by $\mathbb{F}_q$ and the vector space of length i vectors over this field by $\mathbb{F}_q^i$. For vectors $\mathbf{x} \in \mathbb{F}_q^i$ we take the convention of them being column vectors, hence, a vector $\mathbf{x}^T$ is always a row vector.

2.4 Error-Correcting Codes

Error-correcting codes are typically used for reliable delivery of digital information over communication channels, which may introduce errors to the transmitted message due to channel noise. Error-correcting codes add redundancy to messages, which enable error detection and correction at the receiver. The theory of error-correcting codes is in general a deep topic and many algorithms for the encoding and decoding processes have been developed for particular code classes. However, in principle for linear error-correcting codes with short codelengths, encoding and decoding can be done using simple matrix and lookup operations, so their implementation is easy.

In the following, we define and discuss important properties of error-correcting codes. Some example codes are also given, but mostly the codes considered for particular applications will be introduced later in the thesis as the corresponding applications are discussed.

2.4.1 Linear Codes

In this section we recall basic definitions and properties of error-correcting codes, focusing on linear codes, i.e., linear subspaces of $\mathbb{F}_q^n$. The error-correcting properties of a code are defined over the metric, which is used for the given code, and for the purposes of this thesis we consider linear codes over the Hamming metric and the Lee metric. For general properties of error-correcting codes we refer to [7], [21], [34].

Definition 2.13 *A code C is a subset of $\mathbb{F}_q^n$. C is called a linear code if C is a linear subspace of $\mathbb{F}_q^n$.*

The elements of C are length n vectors, which are called codewords. A linear code C of dimension $k \leq n$ is spanned by k linearly independent vectors of C, i.e., every codeword can be written as a linear combination of these k linearly independent vectors.

Definition 2.14 *A matrix $\mathbf{G}$ having as rows any k linearly independent vectors of C is called a generator matrix of the code C.*

If the code has length n and dimension k it is called an (n, k) code.

Define that two column vectors $\mathbf{x}$ and $\mathbf{y}$ are orthogonal if $\mathbf{x}^T\mathbf{y} = 0$. The code C of length n and dimension k can equivalently be specified by listing $n - k$ linearly independent vectors of $C^\perp$, where $C^\perp$ is the subset of $\mathbb{F}_q^n$ consisting of all vectors orthogonal to all vectors of C. Any matrix $\mathbf{H}$ having as rows such $n - k$ linearly independent vectors is called a parity check matrix of C.

Definition 2.15 *Two codes are equivalent if and only if their generator matrices are obtained from each other by column permutations and elementary row operations.*

Since the rows of the generator matrix are linearly independent and span the linear code, i.e., linear subspace of $\mathbb{F}_q^n$, elementary row operations leave the space unchanged. The permutation of columns corresponds to permutation of symbols in the codewords. When these operations are performed for the generator matrix $\mathbf{G}$ of the code C, the resulting code will be only trivially different from C.

A generator matrix $\mathbf{G}$ of the code C is in systematic form if

$$\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$$

$$= \left[\begin{array}{ccccc|cccc} 1 & 0 & 0 & \cdots & 0 & p_{1,1} & p_{1,2} & \cdots & p_{1,n-k} \\ 0 & 1 & 0 & \cdots & 0 & p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ & & & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{k,1} & p_{k,2} & \cdots & p_{k,n-k} \end{array}\right],$$

where $\mathbf{P}$ is called the parity part of the generator matrix. Any generator matrix of C can be put into this form by column permutations and elementary row operations. The resulting generator matrix defines a systematic code, which is equivalent to C. Also, if the generator matrix of C is $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$, it is clear that the parity check matrix $\mathbf{H}$ is of the form $\left[-\mathbf{P}^T|\mathbf{I}_{n-k}\right]$, since

$$\mathbf{G}\mathbf{H}^T = [\mathbf{I}_k\ \mathbf{P}] \left[\begin{array}{c} -\mathbf{P} \\ \mathbf{I}_{n-k} \end{array}\right] = -\mathbf{P} + \mathbf{P} = \mathbf{0}.$$

For binary codes, we may write $\mathbf{H}$ in the form $\left[\mathbf{P}^T|\mathbf{I}_{n-k}\right]$.

The code C encodes an information word $\mathbf{i} = [i_0, i_1, \ldots, i_{k-1}]^T$ to a length n codeword $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]^T$ by matrix multiplication $\mathbf{c}^T = \mathbf{i}^T \cdot \mathbf{G}$. Thus, the code C can be defined as

$$C = \{\mathbf{i}^T\mathbf{G} \mid \mathbf{i} \in \mathbb{F}_q^k\}$$

and equivalently with the parity check matrix $\mathbf{H}$ as

$$C = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}^T\mathbf{H}^T = 0\}.$$

For example, if we have the following generator matrix:

$$\mathbf{G} = \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right],$$

then the parity check matrix H for this code is:

$$\mathbf{H} = \left[\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right],$$

and encoding the information word $\mathbf{i} = [0,1]^T$ we get the corresponding codeword $\mathbf{i}^T\mathbf{G} = [0,1] \cdot \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} = [0,1,1,0,1]$. Here $k = 2$ and $n = 5$.

2.4.2 The Hamming Metric

The vector space $\mathbb{F}_q^n$ can be made into a metric space by defining the distance between two vectors $\mathbf{x}$ and $\mathbf{y}$. The Hamming metric is the most commonly used metric for defining this distance and the properties of the error-correcting code.

Definition 2.16 *The Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ of vectors $\mathbf{x}$ and $\mathbf{y}$ of length n is the number of coordinates where $\mathbf{x}$ and $\mathbf{y}$ differ, i.e., $d_H(\mathbf{x}, \mathbf{y}) = |\{i \mid x_i \neq y_i\}|$.*

Consider the vectors $\mathbf{x} = [0,1,0,0]^T$ and $\mathbf{y} = [1,1,0,0]^T \in \mathbb{F}_2^4$. The vectors differ in the first entry, hence, the Hamming distance is $d_H(\mathbf{x}, \mathbf{y}) = 1$.

Definition 2.17 *The Hamming-weight w_H of a vector $\mathbf{x}$ is $w_H = d_H(\mathbf{0}, \mathbf{x})$.*

Definition 2.18 *A code C is e-error correcting if the minimum Hamming distance between two codewords is $2e + 1$.*

An e-error-correcting code can detect and correct up to e errors in the encoded information word. This means, that if the values of the encoded information word

$\mathbf{i}^T\mathbf{G}$ change in $\leq e$ positions, the decoder of the code will be able to detect and correct the errors, i.e., interpret the received sequence as the correct information word.

Definition 2.19 *A code $C \in \mathbb{F}_q^n$ is called a q-ary τ-covering code of length n if for every word $\mathbf{y} \in \mathbb{F}_q^n$ there is a codeword $\mathbf{x} \in C$ such that the Hamming distance $d_H(\mathbf{x}, \mathbf{y}) \leq \tau$. The smallest such τ is called the covering radius of the code.*

In other words, the covering radius of the code is the smallest τ such that the finite metric space $\mathbf{F}_q^n$ is exhausted by spheres of radius τ around the codewords.

Definition 2.20 *A code is called perfect if it is e-error correcting and its covering radius is e.*

For a perfect code, the entire metric space is filled by the radius e spheres around the codewords, with no overlaps. This means, that every length n sequence can be traced back to a length k information word. However, if more than e errors occur, the interpretation will be incorrect.

An important example of perfect codes in the Hamming metric are the Hamming codes. Binary Hamming codes are a family of $(2^m - 1, 2^m - m - 1)$ codes with parity check matrices consisting of all $2^m - 1$ distinct non-zero m-tuples. Since all of the columns are distinct, no sum of two columns can be the zero vector, and hence the code has minimum distance of ≥ 3. Therefore, Hamming codes are one error correcting. Hamming codes have the covering radius of one, hence they are perfect codes. This means that for each binary vector $\mathbf{v}$ of length $2^m - 1$ there is a unique codeword within distance 1 from $\mathbf{v}$.

For example, the parity check matrix $\mathbf{H}$ of the binary (7,4) Hamming code can be constructed by listing all vectors of length 3 as columns:

$$\mathbf{H} = \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

Therefore, its generator matrix is

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right].$$

Hamming codes are easily defined for vector spaces $\mathbb{F}_q^n$ having parameters $(\frac{q^m-1}{q-1}, \frac{q^m-1}{q-1} - m)$. For each m, there are $(q^m - 1)$ different nonzero vectors, but since the

minimum distance of the code is ≥ 3, the columns of the parity check matrix have to be pairwise linearly independent. Therefore, there are $\frac{q^m-1}{q-1}$ distinct q-ary vectors that we can list as columns of the parity check matrix $\mathbf{H}$.

2.4.3 The Lee Metric

Another metric used for error-correcting codes is the Lee metric. Let us call a Lee-error-correcting code any error-correcting code defined for the Lee metric.

Definition 2.21 *The Lee distance $d_L(\mathbf{x}, \mathbf{y})$ of q-ary vectors $\mathbf{x}$ and $\mathbf{y}$ of length n is the sum $\sum_{i=1}^{n} \min(|x_i - y_i|, q - |x_i - y_i|)$.*

For example, for $q = 5$,

$$d_L(0,1) = 1,\ d_L(0,2) = 2,\ d_L(0,3) = 2 \text{ and } d_L(0,4) = 1.$$

And similarly for vectors $\mathbf{x} = [2,1,3]^T$ and $\mathbf{y} = [1,1,0]^T$, $\mathbf{x}, \mathbf{y} \in \mathbb{F}_5^3$, the Lee distance $d_L(\mathbf{x}, \mathbf{y}) = 1 + 0 + 2 = 3$.

Also, $d_L(\mathbf{a} + \mathbf{k}, \mathbf{b} + \mathbf{k}) = d_L(\mathbf{a}, \mathbf{b})$, i.e., like the Hamming metric, the Lee metric is translation invariant. Notice, that for $q = 2$ and $q = 3$, the Hamming and the Lee metric coincide.

Definition 2.22 *The Lee-weight w_L of a vector $\mathbf{x}$ is $w_L = d_L(\mathbf{0}, \mathbf{x})$.*

An e-Lee-error-correcting code will be able to detect and correct errors, which are at the Lee distance $\leq e$ from the encoded sequence $\mathbf{i}^T\mathbf{G}$, i.e., have Lee-weight $\leq e$.

Perfect codes exist for the Lee metrics also. For example, for any given e, there exists a perfect e-Lee-error-correcting code with $n = 2$ over $\mathbb{F}_q$ such that $q = 2e^2 + 2e + 1$.

2.4.4 Decoding of Linear Codes

When error-correcting codes are used in communication, the underlying assumption is that a digit in a codeword has a small probability to change from one value to another during the transmission over a noisy channel. Thus the most likely original codeword is the one that is closest in the used metric to the received one. Hence the decoding is unique (though not necessarily correct) as long as there is a unique codeword closest to the received vector.

Consider the codewords of an e-error-correcting linear code $\mathbf{C}$. Around each $\mathbf{c} \in \mathbf{C}$ there is a sphere of radius e containing all the words at distance $\leq e$ from $\mathbf{c}$, i.e., all the words that will be decoded into the word c. Hence, the decoding process

can be done by listing all the members of every sphere or more efficiently by using a structure called the standard array.

The standard array uses the coset decomposition of the additive group of $\mathbb{F}_q^n$. Since a linear code is a subspace of $\mathbb{F}_q^n$, it is a subgroup of the additive group of $\mathbb{F}_q^n$ and hence the all-zero word is always a codeword. Thus each vector in the space belongs to exactly one coset of the code. The vectors with minimum weight (i.e., minimum distance from the all-zero codeword) are called coset leaders. It is clear that for an e-error correcting code, all vectors of weight at most e are coset leaders. This follows immediately from the fact that the difference of two vectors in the same coset belongs to the code. A moment of thought shows that if each coset has only one leader then for each element in the space there is a unique closest codeword. Choosing that closest codeword when decoding the received vector corresponds to maximum likelihood decoding [34].

The standard array in Table 2.3 is constructed as follows. Let $\mathbf{0}, \mathbf{c}_2, \ldots, \mathbf{c}_{q^k}$ be the codewords of $\mathbf{C}$, which is a subspace of $\mathbb{F}_q^n$. Begin by listing these words on the first row of the standard array. Then, choose one unused word $\mathbf{v}_1$ at distance $\leq e$ from the all-zero word, i.e., a coset leader, and place it as the first entry of the second row. The rest of the words on the second row will be $\mathbf{v}_1 + \mathbf{c}_2, \ldots, \mathbf{v}_1 + \mathbf{c}_{q^k}$, i.e., the word $\mathbf{v}_1$ translated by each of the codewords. These are the words belonging to the coset of $\mathbf{v}_1$. Then, choose again an unused word at distance $\leq e$ from the all-zero codeword and translate it with each codeword to get the next row corresponding to the next coset. Repeat this until there are no unused coset leaders. If there are still unused words that do not appear in the standard array, draw a horizontal line across the array, and to obtain the next row of the array, choose a word $\mathbf{v}_j$, which is as close as possible to the all-zero word and translate the word $\mathbf{v}_j$ by each of the codewords. Repeat this until there are no unused words in $\mathbb{F}_q^n$.

Table 2.3: The standard array.

$\mathbf{0}$	$\mathbf{c}_2$	$\ldots$	$\mathbf{c}_{q^k}$
$\mathbf{v}_1$	$\mathbf{v}_1 + \mathbf{c}_2$	$\ldots$	$\mathbf{v}_1 + \mathbf{c}_{q^k}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{v}_i$	$\mathbf{v}_i + \mathbf{c}_2$	$\ldots$	$\mathbf{v}_i + \mathbf{c}_{q^k}$
$\mathbf{v}_j$	$\mathbf{v}_j + \mathbf{c}_2$	$\ldots$	$\mathbf{v}_j + \mathbf{c}_{q^k}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

In terms of the standard array, a decoder would assign a received word having at most e errors (i.e., appearing above the horizontal line), into the codeword, which lies above it in the standard array. Words below the horizontal line are either assigned

to a codeword similarly as the words at distance e from the codeword, or the decoder refuses to decode such a word indicating more than e errors in the received word. For perfect codes, there are no words under the horizontal line, since the spheres around the codewords fill the entire metric space.

The coset leaders can be used to construct a fast decoding strategy by introducing the syndrome $\mathbf{s}$ of a word $\mathbf{v} \in \mathbb{F}_q^n$, which is defined as $\mathbf{s}^T = \mathbf{v}^T\mathbf{H}^T$. It can be shown that all vectors in the same coset have the same syndrome, which is unique to that coset [34]. Hence, for efficient decoding, we only need to list the coset leaders and the syndromes. Then, for each received word, it is enough to compute the syndrome and find the corresponding coset leader. Subtracting the coset leader from the received word will correct the error.

3. FAULT-TOLERANCE IN LOGIC CIRCUITS

Fault-tolerance is an important topic in logic design and there exist several techniques for providing tolerance against hardware component failures. In some cases, for example in military and aerospace computing, fault-tolerance is critical, and it is often desirable in other applications. Even when using high-quality components and the best design techniques, system failures can still occur, and therefore techniques for providing fault-tolerance are necessary. The first ideas for fault-tolerant logic were given by von Neumann [35] and Shannon and Moore [24] already in the 1950s. In the work of Shannon and Moore it was shown that arbitrarily reliable circuits can be built from unreliable components when the number of these unreliable components is sufficiently large.

Failures in digital systems may be due to numerous reasons, and we begin by an overview of the most common types of faults that may result in errors. Next, the basic strategies for increasing fault-tolerance are discussed in Section 3.2. One of the most well-known techniques against hardware component failures is triple modular redundancy (TMR), which is explained in Section 3.3. The TMR technique has been studied and improvements have been presented in several papers. There exist also methods for increasing fault-tolerance, which utilize more complicated error-correcting codes, and some of these techniques are reviewed in Sections 3.4-3.6.

3.1 Faults in Digital Systems

In digital systems, a system failure means that an element is unable to function due to errors in the element or in its environment, the errors being caused by various faults [26]. Faults are physical defects that can be caused by numerous reasons, e.g. design errors, damage or external disturbances. Faults are manifested by errors, thus a fault may change the value of the signal. A fault does not necessarily result in an error, which gives meaning for the term fault-tolerance. The purpose of fault-tolerant design is to ensure that a system can perform its intended function even in the presence of a given number of faults.

Such faults, which change the value of the signal, e.g. from a 0 to a 1 or vice versa, are called logical faults [19]. Other types of faults are referred to as nonlogical, which include, for example, the malfunction of the clock signal or a power failure. Logical faults are the type of faults that are of interest for the fault-tolerance methods

discussed in this thesis, and they can be further characterized by their value, extent and duration [19]:

1. Value: Logical faults can cause fixed or varying erroneous logical values.
2. Extent: The effect of a logical fault can be local or distributed.
3. Duration: A logical fault can be either permanent or temporary.

Different types of faults are described by fault models. The most common model used for logical faults is the single stuck-at fault. A stuck-at-fault is a fault, in which one of the inputs or the output of a gate is fixed to some value, e.g. in switching circuits to a 0 or a 1 [19]. The stuck-at-fault model is used for modeling the most common types of physical defects in circuits, e.g. shorts, which may cause damage to the circuit due to very low resistance in the circuit, and opens, in which the path of the current gets broken. The stuck-at-model can also be used for representing multiple faults in a circuit, when there are multiple stuck-at-faults in the circuit at the same time.

When signal lines in a circuit get accidentally connected to each other, it results in a permanent fault called a bridging fault [19]. Bridging faults are connected to stuck-at-faults, since, depending on the technology, a bridging fault may manifest as a stuck-at-fault. It may also cause a circuit to oscillate. The bridging fault model is more applicable when the line widths are small.

A defect in a circuit may also be small enough not to alter the logic function, which the circuit realizes, but will only cause the circuit to fail to meet its timing specifications [19]. Such defects delay the transition of a signal on a line to the correct value, and are modeled by delay faults.

3.2 Fault-Tolerance Strategies

The foundation for fault-tolerance is in redundancy. Redundancy may be introduced to hardware, software, information and computations, and the amount of redundancy depends on the applied fault-tolerance technique. Fault-tolerance strategies include one or more of the following elements [26]:

1. Masking, i.e., dynamic correction of generated errors.
2. Detection of an error, i.e., a symptom of a fault.
3. Containment, which means the prevention of error propagation.
4. Diagnosis, i.e., identification of the faulty module.
5. Repair or reconfiguration of the faulty component by replacement, elimination or bypassing it.

6. Recovery, i.e., correction of the system to an acceptable state.

Masking, detection and correction are most important with respect to the topic of this thesis, and several fault-tolerance techniques carry out one or more of these three. Error detection, i.e., the ability to tell the incorrect values apart from the correct ones, is the easiest of these to implement. An example of an error detection scheme is simple parity checks in buses, memory and registers. Masking and correcting errors is more difficult, and often requires a lot of redundancy. This redundancy can be obtained, for example, by copying modules, which is the basic principle in TMR technique (Section 3.3).

Error-correcting codes are an effective way of obtaining redundancy, and they often require less redundancy than other error detection and correction schemes. Their theory is well developed and exploited in numerous applications. In logic circuits, error detection and correction is usually implemented in special decoding circuits [26]. The performance of the error-correcting scheme depends on the properties of the code, and better error-correction ability usually requires more redundancy and therefore, more physical space on circuits.

3.3 Triple Modular Redundancy

The TMR technique was first introduced by von Neumann in [35], where he proposed a configuration of independently computed copies of a signal and a restoring organ in between logical operations. The basic idea of the TMR technique is to triplicate the modules and then use a majority voter to decide the output of the whole system. This way, if one of the modules produces an incorrect output, the majority vote will still guarantee a correct output for the entire system. The TMR technique can be generalized to N-modular redundancy (NMR), where there are in total N modules and a voter, which decides the output based on the outputs of the modules.

The NMR technique can be described in terms of a $(r, 1)$ repetition code, where $r = N$. Repetition code is the simplest linear code. In an $(r, 1)$ binary repetition code a digit is encoded as a sequence of r repetitions of the digit itself, and the decoding is done by majority-vote decoding. For example, in the binary $(3, 1)$ repetition code, which relates to the TMR technique, a 0 is encoded as the sequence 000, and if the decoder receives either 000, 001, 010 or 100, it decodes the sequence as a 0 by majority-vote decoding. Therefore, the $(3, 1)$ repetition code is one error correcting.

In the TMR technique, usually an entire module realizing some logical operation is encoded, i.e., triplicated, and a single voter is placed after the three modules (Figure 3.1) or, more typically, a voter is placed after each module. These voters together decode the output to a single output value. This output can be a symbol, i.e., a sequence of bits, and therefore the TMR network can correct single symbol

errors. In Figure 3.2a is a network of modules and in Figure 3.2b is a typical TMR version of this network, where the circles correspond to voters. The inputs are i_1, i_2, i_3 and outputs o_1, o_2.

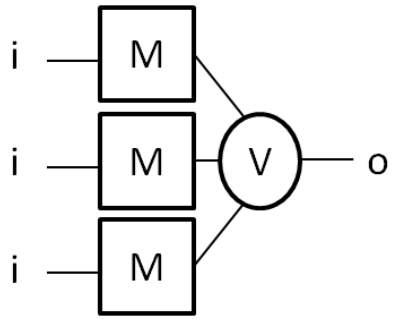


Figure 3.1: The simplest TMR structure, with input i and output o.

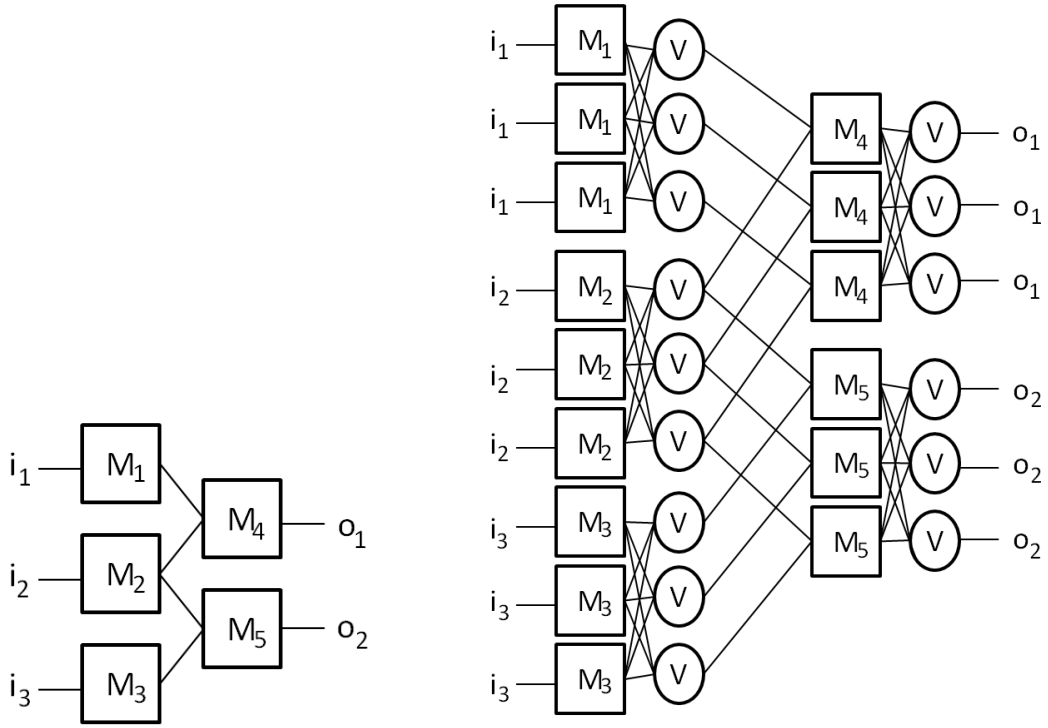


Figure 3.2: A network of modules (a) and its TMR version (b) with three inputs i_1, i_2, i_3 and two outputs o_1, o_2.

In a TMR computer architecture, a module consisting of a memory and a processor is triplicated, and a voter is added after each triplicated module. This allows correction of single symbol errors, but bit errors distributed over the modules cannot be corrected. In [17], an improvement for the TMR technique was proposed, where

the memory part of the modules is organized in a bit-sliced way, i.e., from modules of smaller bit width, which increases the amount of hardware in the configuration, but can tolerate arbitrary bit-slice failures. It can also tolerate the failure of arbitrary bit-slices even if one of the modules is disabled. This idea of combined symbol- and bit-error-correcting in a computer architecture originates from the (N, K) concept, which is discussed in Section 3.5.

The TMR technique is the most widely used fault-tolerance technique. It has been used in highly critical applications, e.g. in space technique [11], as well as in some ECC memory, where ECC refers to error-correcting codes, although ECC memory more typically uses Hamming codes [14].

3.4 Self-Checking Circuits

Self-checking circuits utilize the theory of error-correcting codes and have built-in error detection capability. They are multi-output circuits, which produce an output vector, from which the possible faults in the circuit can be detected. Formally, the output of a self-checking circuit is a vector $\mathbf{Y}(\mathbf{X}, f)$ which is a function of the input vector $\mathbf{X}$ and the fault f in the circuit [36]. The inputs and outputs are codewords, and the use of different error-correcting codes in these circuits has been studied by several authors. When no faults occur in a self-checking circuit, the output vector is a codeword, but a fault should result in a non-codeword output vector (detectable error). However, it may happen that faults in the circuit result in an incorrect codeword, in which case the fault will result in an undetectable error.

Self-checking circuits are divided to fault secure circuits, self-testing circuits or totally self-checking (TSC) circuits [26]. In fault secure circuits, a correct input codeword never causes an incorrect output codeword for a specified fault in the circuit, i.e., no undetectable errors for correct input codewords can occur. However, a fault might not result in a detectable error either. For self-testing circuits, if a fault occurs in the circuit, the output is a non-codeword for at least one input codeword. This means that if all input codewords occur in the operation of the circuit, a fault will be detected by at least one of these input codewords. TSC circuits have the properties of both self-testing and fault secure circuits, since a fault in the circuit, in principle, cannot cause an error in the outputs without detection of the fault [27]. Therefore, an incorrect input cannot result in a correct output and at least one correct input will detect possible faults in the circuit.

TSC circuits are the most important class of self-checking circuits, and the above properties guarantee that if the output is a codeword, it is safe to assume that it is correct, and on the other hand, if there is a fault in the circuit, it will be detected at some point. However, it may happen that faults are not detected in the order of their appearance [27]. This might be the case if the first fault is detectable only by

a particular input codeword. Also, implementing the self-testing properties in an economical way has been problematic.

The basic implementation of a TSC network is introducing a TSC checker into the network [27]. The TSC checker is a circuit, which takes as inputs the outputs of the TSC network and is designed to detect errors in the error-correcting code used in the network. Hence, the TSC checker is an additional piece of hardware, which handles the error detection of the designed network. Possible faults can be detected from the outputs of the checker. However, there is no information on whether the fault occurred in the TSC network or in the checker.

3.5 The (N, K) Concept

The (N, K) concept was introduced by Krol in [18] as a new fault-tolerant computer architecture based on a distributed implementation of a symbol-error-correcting code. It is essentially a generalization of the TMR technique in which different coding schemes are applied to memory data and processor data. In a TMR computer architecture, both the memory data and the processor data are encoded using the $(3, 1)$ repetition code, but with the (N, K) concept, the amount of additional memory hardware can be reduced. In [18], the concept was explained in more detail for the case where $N = 4$ and $K = 2$, in which the fault-tolerant computer architecture is designed using a specific symbol- and bit-error-correcting code. The term symbol refers to a sequence of bits, so within a faulty symbol, any number of bits can be incorrect.

The (N, K) concept makes it possible to choose a ratio between memory and processor redundancy and therefore makes it possible to optimize the total amount of redundancy. In [18], the TMR technique was described as the $(3, 1)$ concept, where the computer architecture consists of three identical modules, each having memory, a processor and a voter. All data in the system is triplicated, and voters mask the possible failures in a single module. In the (N, K) concept, the processor data is similarly encoded into a $(N, 1)$ symbol-error-correcting code, but the memory part is encoded into an (N, K) symbol-error-correcting code, where each of the N modules contains one symbol of the code word. However, the symbol size in the (N, K) code is K times smaller than the symbol size of the $(N, 1)$ code, which means that less redundancy is introduced when using the (N, K) concept.

The concept is easier to understand by means of an example, e.g. the $(4, 2)$ concept. In this case, the computer architecture consist of four modules, each having memory and a processor. The processors are identical in all modules, but the memory part of each module has a wordlength of half a data word, i.e., the memory in the $(4, 2)$ concept is only doubled. The memory in each module is protected by a symbol- and bit-error-correcting code, which can correct single symbol errors and

double bit errors even if the bit errors occur in different modules. If symbol errors occur in more than one of the modules, the code can no longer correct the errors. In Figure 3.3 is the layout of the modules, in which the wordlengths in each line is written on the transfer lines. The boxes labeled E_1, E_2, E_3, E_4 together form the encoder of the $(4, 2)$ symbol-error-correcting code. This way, data is encoded when it is transmitted from the processors to the memories, but the encoding is distributed over the modules to reduce redundancy. When the data is transmitted from the memories to the processors, each module receives the complete codeword and the decoders mask the faulty symbols.

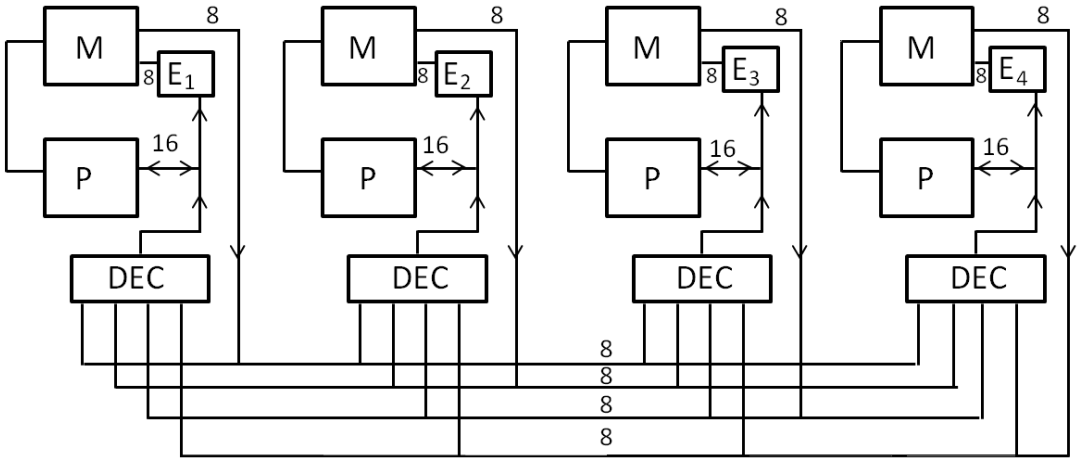


Figure 3.3: The layout of the $(4, 2)$ concept fault-tolerant computer architecture.

Since the (N, K) concept discusses a complete fault-tolerant computer architecture, it is not as such applicable to circuit design. It is, in principle, a generalization of the TMR technique, which enables the usage of more complicated error-correcting codes and the choice in the ratio between the memory and processor redundancy. The advantage of the (N, K) concept compared to the TMR technique is the possibility of minimizing the total amount of hardware.

3.6 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were introduced already in the 1960s by Gallager in [15], but they were rediscovered for applications in the 1990s. LDPC codes are a class of codes, which are specified by sparse matrices, i.e., matrices containing mostly zeros. They have been popular in many communication applications, since the encoding matrices have low density and decoding can be done iteratively in an easy manner. Here we briefly review some selected applications of LDPC codes in fault-tolerant logic, in which the use of LDPC codes is motivated by their high error detecting and correcting ability, and because the resulting encoding, decoding

and checker circuits are sparse. In [10] and [16], LDPC codes are considered for error correction in fault-tolerant memory, and in [22] the application of LDPC codes in fault-tolerant finite field multipliers is introduced.

The application of LDPC codes in nanoscale ECC memory in [16] is interesting, since it considers modern nanoscale logic. A particular type of LDPC codes is used in the application, namely Euclidean Geometry LDPC codes, see [32], since the decoding process for these codes can be implemented with multistep majority decoders, thus making the decoding fast and giving low latency, i.e., time delay, for memory operations. In [16], the possibility of faults in the encoder and checker circuits is taken into account, since they are also fabricated out of nanoscale components, which are prone to faults. An efficient way of fabricating the encoder and decoder circuits using nanowire-based programmable logic arrays (PLAs) is proposed in [16], and the error detection and correction capacities of the resulting ECC nanomemories are analyzed.

In [10], LDPC codes are used for a fault-tolerant memory architecture. The motivation behind the application is again in the decrease of transistor sizes, which increases the unreliability of the components. In particular, the authors consider unreliable components, which are subject to transient errors, i.e., errors, which appear at particular time steps. The idea is to build a reliable memory entirely of unreliable components using LDPC codes. The existence of such reliable memories is proved in [10], and it is also shown that the resulting architecture will have fairly low redundancy.

The application of LDPC codes in [22] is motivated by fault related attacks in cryptographic hardware, which is used, for example, for digital signature and identification schemes. It has been shown that attackers can inject faults into the hardware causing incorrect outputs, which expose the digital signatures. Cryptography applications use finite field arithmetic, and in [22], a method for designing error-correcting multiplier circuits for finite fields is proposed, in which LDPC codes are used due to their reduced decoding complexity. The resulting applications are shown to have significantly less redundancy than the traditional TMR technique, but additional delay is introduced to the circuits due to encoding and decoding procedures.

4. ERROR-CORRECTING DECISION DIAGRAMS

Many of the existing fault-tolerance techniques, e.g. the TMR technique (Section 3.3), are based on repetition of logic modules and additional checking circuitry. In modern logic circuits, transistors are becoming smaller and smaller, and even atomic-scale imperfections and variations within each transistor can result in circuit failures. Therefore, in addition to testing and fault detection procedures, it is important to find systematic ways of introducing fault-tolerance already into the representations of switching functions. When redundancy is introduced to the representations of functions, no additional voters or checker circuits are required. This is the motivation behind combining the theory of error-correcting codes and decision diagrams, since decision diagrams are an efficient way of representing functions. The original method presented in this chapter introduces robustness to the representations of functions, and due to the properties of decision diagrams, the information on the complexity and layout of the resulting circuits is contained in these robust representations. Each robust diagram is specified by an error-correcting code, and the properties of the given code affect the complexity and layout of the diagram. The procedure of constructing the robust diagram is analogous to the decoding process of the code, which for linear codes is, in principle, based on simple matrix and lookup operations.

First, some introduction to the subject is provided in Section 4.1 before moving onto definitions and details. The definition of error-correcting decision diagrams is given in Section 4.2, and the step by step procedure of constructing such a diagram is described in Section 4.3. In Section 4.4, several examples of error-correcting decision diagrams in both binary and multiple-valued logic are given.

4.1 Introduction to Error-Correcting Decision Diagrams

Consider the TMR technique described in Section 3.3. If instead of logic modules, we simply consider variables, then TMR realizes the majority function f_4 of three variables, for which the truth vector is given in Table 4.1. On the other hand, the majority function f_4 has a direct correspondence to the binary $(3, 1)$ repetition code, since the function has the values 0 and 1, when the received length 3 vector is decoded to 0 and 1, respectively, by the decoding rule of the repetition code.

The majority function f_4 can be represented by its BDD, which is shown in Figure

Table 4.1: The truth-table of the majority function f_4 of 3 variables.

$x_0x_1x_2$	$f_4(x_0, x_1, x_2)$
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

4.1. There is some similarity between the structure of the BDD and the structure of a logic module with TMR (Figure 3.1), i.e., the four nodes correspond to the triplicated units and a voter. Also, due to the correspondence between the majority function f_4 and the $(3, 1)$ repetition code, this diagram can be thought of as the BDD representing the decoding rule of the $(3, 1)$ repetition code, where by following the edges corresponding to the received sequence, the original information word is found in the terminal node. In other words, the received vector is the input and the output is given by the decoding rule of the $(3, 1)$ repetition code. Hence, the BDD in Figure 4.1 is the simplest error-correcting decision diagram.

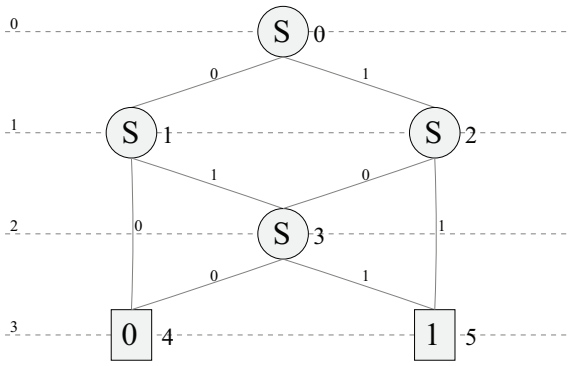


Figure 4.1: The BDD of the majority function f_4.

The above diagram is a robust representation of the 1-variable function f_5, which is 0 when the input is 0, and 1 when the input is 1. The binary decision diagram of the function f_5 consists of just a single non-terminal node and two terminal nodes. The basic idea of how the decision diagram in Figure 4.1 is obtained from f_5 is to map the function f_5 to the majority function f_4 given in Table 4.1. The mapping is done by assigning such length 3 vectors to 0 (1), which would be decoded to 0 (1) by the decoding rule of the $(3, 1)$ repetition code. Then, the BDD of the majority function f_4 will be the error-correcting BDD of the function f_5. Hence, the BDD in

Figure 4.1 is a robust version of a single decision node.

The idea can be generalized to arbitrary functions and codes. The procedure of generating error-correcting decision diagrams follows the decoding rule of the code, and the error-correcting properties of the diagrams depend on the given code. Using an e-error-correcting code leads to an error-correcting decision diagram, which corrects e decision errors. If the utilized code is a linear (n, k) code, it is a subspace of the vector space $\mathbb{F}_q^n$, and the paths of an error-correcting decision diagram correspond to the elements of the additive group of $\mathbb{F}_q^n$. The paths corresponding to elements belonging to cosets having coset leaders at distance $\leq e$ from the all-zero codeword lead to the correct output value.

Now, if the error-correcting capability of a given code is e, then in the error-correcting decision diagram generated with the given code, even if a wrong decision is made in up to e nodes, we will still end up in the correct terminal node. For example, when $e = 1$, if we follow the edges of the diagram towards the terminal node, we may take a wrong turn once on the path that we are following, and we will still end up in the correct terminal node. This is illustrated in Figure 4.2, where following both the edges labeled 0 and 1 after the node on level 1 results in the same output value. Notice, that the edge from the left-hand-side node on level 1 to the terminal node labeled with 0 also includes one change in the input value.

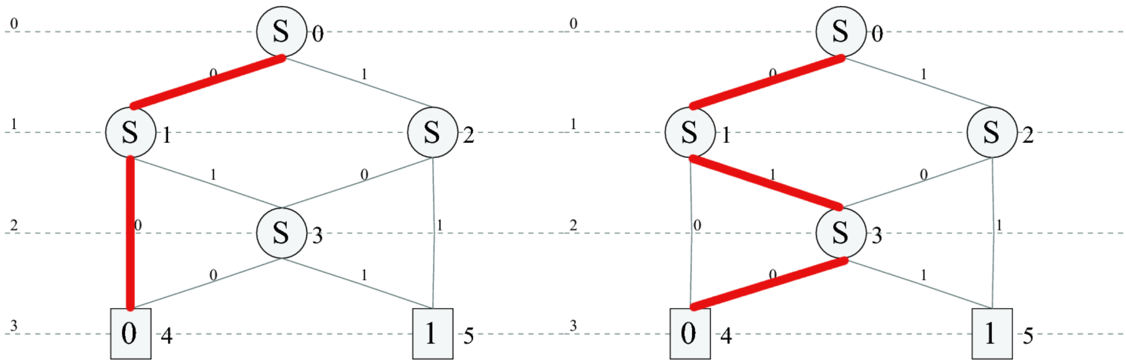


Figure 4.2: Illustration of how one faulty node on a path does not affect the output value.

The concept of error-correcting decision diagrams is a new method for introducing fault-tolerance in logic design. In this thesis, the method is discussed on a theoretical level, but once the decision diagrams are generated, their implementation is straightforward with a suitable technology. The ideas introduced in this thesis have also resulted in two conference papers [4], [5], where error-correcting decision diagrams and their performance is discussed.

4.2 Formal Definition of Error-Correcting Decision Diagrams

The definition of error-correcting decision diagrams is given in terms of linear codes. However, the idea can be generalized to apply for any coding schemes. This possibility is briefly discussed in Section 6. In the following definition, we denote by $d(\mathbf{x}, \mathbf{y})$ the distance between the vectors $\mathbf{x}, \mathbf{y}$. This distance is assumed to be a metric.

Definition 4.1 *Let* $\mathbf{G}$ *be the generator matrix of a linear* e*-error-correcting* (n, k) *code,* $f = f(x_0, x_1, \ldots, x_{k-1})$ *a function, where* $[x_0, x_1, \ldots, x_{k-1}]^T \in \mathbb{F}_q^k$ *and* $g = g(y_0, y_1, \ldots, y_{n-1})$ *a function, where* $[y_0, y_1, \ldots, y_{n-1}]^T \in \mathbb{F}_q^n$*. The error-correcting decision diagram of* f *is the decision diagram of* g*, where* g *is defined as*

$$g(\mathbf{y}) = \begin{cases} f(\mathbf{x}) & \text{if there is } \mathbf{x} \in \mathbb{F}_q^k \text{ such that } d(\mathbf{y}^T, \mathbf{x}^T\mathbf{G}) \leq e \\ * & \text{otherwise,} \end{cases} \tag{4.1}$$

and the value $*$ *can be chosen arbitrarily.*

In other words, each $\mathbf{x}^T\mathbf{G}$ and the vectors $\mathbf{y}^T$, where $\mathbf{y} \in \mathbb{F}_q^n$, within distance e from $\mathbf{x}^T\mathbf{G}$ are assigned to the value $f(\mathbf{x})$. The vectors $\mathbf{y} \in \mathbb{F}_q^n$ at distance $> e$ from all the codewords are assigned to the label $*$. The symbol $*$ can be some arbitrary value, which can be defined in a suitable way. It can have a value $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{F}_q^k$, or it can have some other value. The function now behaves as the decoding rule of the code having the generator matrix $\mathbf{G}$, i.e., the vectors of $\mathbb{F}_q^n$ within distance e from a codeword $\mathbf{x}^T\mathbf{G}$ are interpreted as the codeword itself when determining the function value. This is just the decoding process, where each received n-ary sequence is interpreted as the codeword within distance e from the received sequence. If the label $*$ is obtained, then more than e decision errors have been made indicating at least $e + 1$ faults in the corresponding circuit.

Definition 4.2 *Given a linear* e*-error-correcting* (n, k) *code, the general error-correcting* q*-ary decision diagram of* k*-variable functions is the error-correcting decision diagram of* $f = f(x_0, x_1, \ldots, x_{k-1})$*, where the function values as* $(x_0, x_1, \ldots, x_{k-1})$ *runs through the domain* $\mathbb{F}_q^k$ *of* f *are left unspecified.*

The error-correcting decision diagram of a particular $f : \mathbb{F}_q^k \to \mathbb{F}_q$ can be derived from the general error-correcting q-ary decision diagram of k-variable functions by assigning the values of the function to the terminal nodes of the error-correcting decision diagram and reducing with respect to those values.

4.3 Constructing Error-Correcting Decision Diagrams

Suppose we wish to have an error-correcting decision diagram for arbitrary q-ary functions of k variables, i.e., we want to construct the general error-correcting q-ary

decision diagram of k-variable functions. The procedure begins by determining a suitable q-ary linear (n, k) code, which corrects e errors. After selecting the code, the function $f = f(x_0, x_1, \ldots, x_{k-1})$, where $[x_0, x_1, \ldots, x_{k-1}]^T \in \mathbb{F}_q^k$, is mapped to the function $g = g(y_0, y_1, \ldots, y_{n-1})$ of larger domain, where $[y_0, y_1, \ldots, y_{n-1}]^T \in \mathbb{F}_q^n$. This mapping is done by equation (4.1) using the specified metric.

The next step is to construct the multi-terminal decision tree having q^n terminal nodes for the function g. Then, the obtained tree is reduced to an MTDD. After reducing, we have a diagram with $q^k + 1$ terminal nodes labeled by $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{F}_q^k$, and $*$. This diagram can correct e decision errors, since the correct function value is obtained even if a decision error occurs in $\leq e$ nodes of the diagram.

From the obtained reduced decision diagram we get a robust decision diagram of a particular function $f : \mathbb{F}_q^k \to \mathbb{F}_q$ by replacing the labels $f(\mathbf{x})$ by the actual values of the function f and reducing the diagram with respect to those values. This diagram will then give a robust layout for a circuit realizing the desired q-ary function of k variables.

The step by step method of constructing error-correcting decision diagrams goes as follows. The last step is only for the case where we wish to obtain the error-correcting decision diagram for a particular function.

1. Multiply the generator matrix $\mathbf{G}$ of the desired (n, k) code by each vector $\mathbf{x}_i^T$, where $\mathbf{x}_i \in \mathbb{F}_q^k$ to obtain $\mathbf{c}_i \in \mathbb{F}_q^n$.

2. For each $\mathbf{c}_i$, list all the vectors $\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, \ldots, \mathbf{y}_{i,u}$ of $\mathbb{F}_q^n$ within distance e from each $\mathbf{c}_i$.

3. To obtain the function g, assign the value $f(\mathbf{x}_i)$ to each $\mathbf{c}_i$ and to the corresponding set of $\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, \ldots, \mathbf{y}_{i,u}$.

4. Map each $\mathbf{y} \in \mathbb{F}_q^n$ at distance $> e$ from all the codewords $\mathbf{c}_i$ to $*$.

5. Construct a MTDD for the function g and reduce it.

6. To obtain the robust decision diagram of a specific function f, assign the values of f into the terminal nodes of the MTDD of the function g and reduce.

In terms of the standard array, the set of vectors $\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, \ldots, \mathbf{y}_{i,u}$ in step 2 corresponds to the vectors listed directly below the codeword $\mathbf{c}_i$, but above the horizontal line. The vectors of the standard array lying below the horizontal line correspond to the paths of the error-correcting decision diagram leading to the node labeled with the symbol $*$. If a coset below the horizontal line in the standard array has a unique coset leader, the elements of that coset could be assigned to the values $f(\mathbf{x}_i)$ corresponding codewords $\mathbf{c}_i$ above them in the standard array to obtain

maximum likelihood decoding. Only the elements in cosets having no unique coset leader would then be assigned to the value $*$.

4.4 Examples

For better understanding of the concept of error-correcting decision diagrams, examples in both binary and multiple-valued logic must be provided. In this section, we give some examples with fairly small values for both q and k, since the number of nodes increases rapidly, as these parameters become larger. For binary functions, we consider examples in the Hamming metric, and for multiple-valued functions, both the Hamming and the Lee metric are considered.

The examples have been generated as follows. A suitable linear (n, k) code was selected for each example, and for obtaining the mapping of the function f to the function g, a script was written in MATLAB [1]. The decision diagrams of the resulting functions were generated using the XML-based framework, which was introduced in [30].

4.4.1 Binary $(5, 2)$ Code

The first example considers a non-perfect $(5, 2)$ code for constructing a general error-correcting MTBDD for 2-variable binary functions. Since the given code is not perfect, the resulting robust diagram will have a terminal node with the label $*$.

Let C be a binary $(5, 2)$ code defined by the generator matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

The parity check matrix $\mathbf{H}$ of the code C is then

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Since the sum of no two columns of $\mathbf{H}$ is zero, the code has minimum distance 3 and corrects 1-bit errors.

Using C, we can construct the general error-correcting decision diagram for binary 2-variable functions. Take all $\mathbf{x} \in \mathbb{F}_2^2$ and for each, compute the codeword $\mathbf{c}^T = \mathbf{x}^T \cdot \mathbf{G}$, where $\mathbf{G}$ is the generator matrix of C. To obtain the function $g(\mathbf{y})$, map each of the codewords to the label $f(\mathbf{x})$ as in equation (4.1). For example, since $[0, 1] \cdot \mathbf{G} = [0, 1, 1, 0, 1]$, the codeword $[0, 1, 1, 0, 1]^T$ is mapped to $f(0, 1)$. Then, for each obtained codeword $\mathbf{c}$, list all the vectors $\mathbf{y} \in \mathbb{F}_2^5$ within distance 1 from $\mathbf{c}$, and

map the vectors to the corresponding $f(\mathbf{x})$, where $\mathbf{c}^T = \mathbf{x}^T\mathbf{G}$.

For example,

$$g(1,1,1,0,1) = g(0,0,1,0,1) = g(0,1,0,0,1)$$
$$= g(0,1,1,1,1) = g(0,1,1,0,0) = g(0,1,1,0,1) = f(0,1).$$

The vectors $\mathbf{y} \in \mathbb{F}_2^5$ at distance > 1 from all the codewords are mapped to $*$.

Next, the multi-terminal binary decision tree for the function g is constructed and reduced to obtain the general error-correcting MTBDD for 2-variable functions (Figure 4.3). The terminal nodes are labeled $f(0,0), \ldots, f(1,1)$ and $*$. To get the MTBDD of a particular binary function, the labels $f(\mathbf{x})$ can be replaced by the actual values of the function at $f(\mathbf{x})$ and the general diagram should then be reduced with respect to these values.

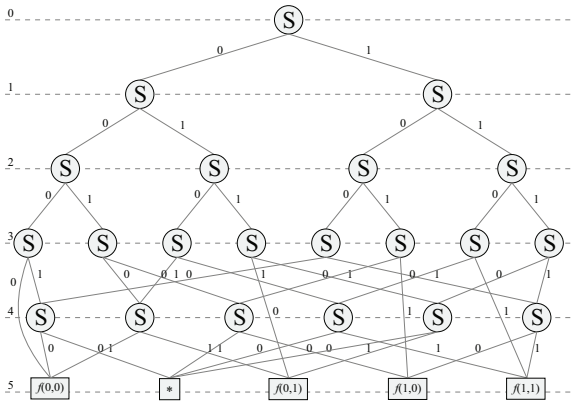


Figure 4.3: A robust MTBDD for 2-variable functions using the $(5,2)$ code.

For example, to get the MTBDD of the 2-variable binary function f_6 defined in Table 4.2, assign the value 0 to the terminal nodes labeled $f(0,0)$ and $f(1,0)$, and the value 1 to the terminal nodes labeled $f(0,1)$ and $f(1,1)$. Then, reducing the obtained diagram we obtain the robust diagram for f_6 (Figure 4.4).

Table 4.2: The truth-table of the function f_6.

x_0x_1	$f_6(x_0, x_1)$
00	0
01	1
10	0
11	1

In the error-correcting decision diagram in Figure 4.4, obtaining the output $*$ indicates at least 2 decision errors. However, a decision error in two nodes is not always detectable, since some of such errors change the codeword into such a word, which

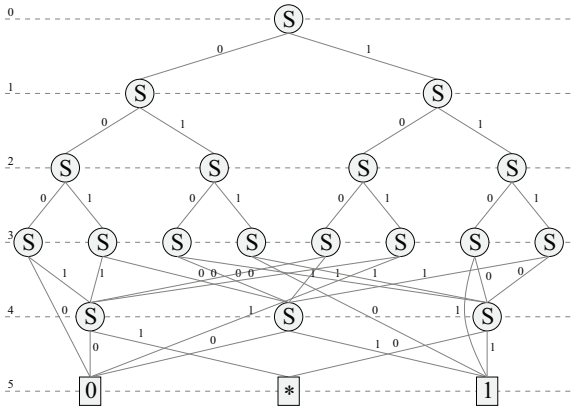


Figure 4.4: A robust MTBDD for the function f_6.

is at distance 1 from some other codeword. For example, the word $[1,1,1,1,1]^T$ is at distance 2 from the codeword $[0,1,1,0,1]^T$, but does not output $*$, since it is at distance 1 from the codeword $[1,1,0,1,1]^T$.

Notice that the function $f_6 = x_1$, which means that it is the identity function of a single variable. Therefore, the diagram in Figure 4.4 can be seen as a competitor for the robust diagram in Figure 4.1, which is generated using the $(3,1)$ repetition code. The diagram in Figure 4.4 shows how the properties of the code affect the resulting diagram, since it represents a simple function but has significantly higher complexity than, e.g. the diagram in Figure 4.1, where the utilized code is simpler.

4.4.2 Hamming $(7,4)$ Code

In the case of binary Hamming codes, the parameters of the code are $n = 2^m - 1$ and $k = 2^m - m - 1$. Therefore, for each binary function with $k = 2^m - m - 1$ variables, we can find a Hamming code and construct the robust BDD using this code. The properties of the Hamming code guarantee that by following the BDD, the correct function value is obtained even if one decision error occurs during the determination of the function value.

Consider the $(7,4)$ Hamming code having the following (non-systematic) generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Using this code we may produce a general error-correcting decision diagram for binary 4-variable functions. First, we need to find the length 7 codewords, which are obtained by multiplying at the left the above generator matrix $\mathbf{G}$ by each $\mathbf{x}^T$, where $\mathbf{x} \in \mathbb{F}_2^4$.

Next, each obtained codeword is mapped to the corresponding symbolic value $f(\mathbf{x})$, and for each codeword, the vectors of length 7 within distance 1 from that codeword are also mapped to the value $f(\mathbf{x})$. Now, we have obtained a function g for which we can generate a MTBDD, which will have in total $2^4 = 16$ terminal nodes (Figure 4.5).

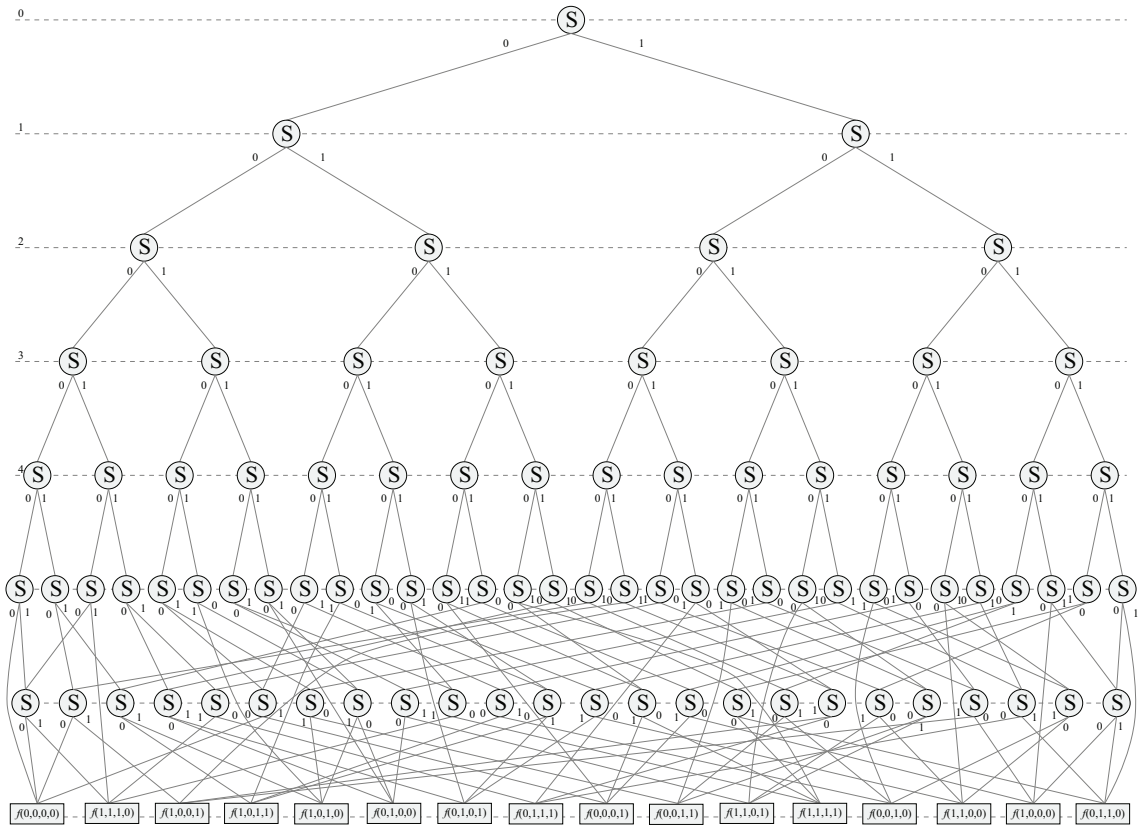


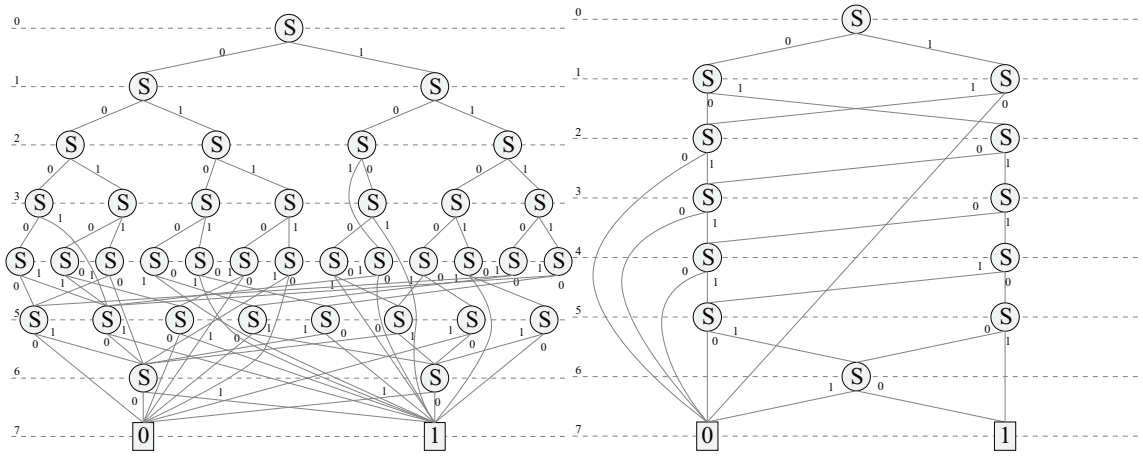
Figure 4.5: The robust MTBDD for 4-variable functions using Hamming $(7, 4)$ code.

Consider the functions $f_7 = x_0x_1 \oplus x_0x_2 \oplus x_0x_3 \oplus x_2x_2 \oplus x_1x_3 \oplus x_2x_3$, where $\oplus$ denotes the exclusive OR, and $f_8 = x_0x_1x_2x_3$. The error-correcting decision diagrams of f_7 and f_8 can be obtained from the diagram in Figure 4.5 by assigning their function values to the terminal nodes and reducing with respect to those values. The resulting diagrams are shown in Figure 4.6.

The two diagrams in Figure 4.6 have a significantly reduced number of nodes than the general MTBDD of 4-variable functions.

4.4.3 Shortened Hamming Code

By shortening an existing linear code, it is possible to form a new linear code, which has some of the properties of the original code. The shortening process is usually done by taking the subspace of the chosen (n, k) code consisting of all codewords, which begin by 0. The 0 is then deleted from the beginning, giving a new linear



(a) $f_7 = x_0x_1 \oplus x_0x_2 \oplus x_0x_3 \oplus x_2x_2 \oplus x_1x_3 \oplus x_2x_3$ (b) $f_8 = x_0x_1x_2x_3$

Figure 4.6: Error-correcting decision diagrams of f_7 and f_8 generated using the Hamming $(7, 4)$ code.

code of parameters $(n-1, k-1)$. For example, by shortening the Hamming $(7, 4)$ code we get a $(6, 3)$ code, which has the same minimum distance and a generator matrix

$$\mathbf{G} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right].$$

However, this code is not perfect.

Consider the full adder of 2 variables, having the truth-table given in Table 4.3. With the carry in, it is a 3-variable function, which has two outputs (Figure 4.7). We can construct a robust decision diagram for the full adder by using the shortened Hamming code of parameters $(6, 3)$.

Table 4.3: The truth-table of the full adder of 2 variables.

ABC_i	C_o	S
000	0	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	1	1

The two-output function is mapped into a function of a larger domain by multiplying at the left the generator matrix of the $(6, 3)$ code by the vectors of $\mathbb{F}_2^3$. The shared MTBDD of the new function is then constructed, assigning two values to

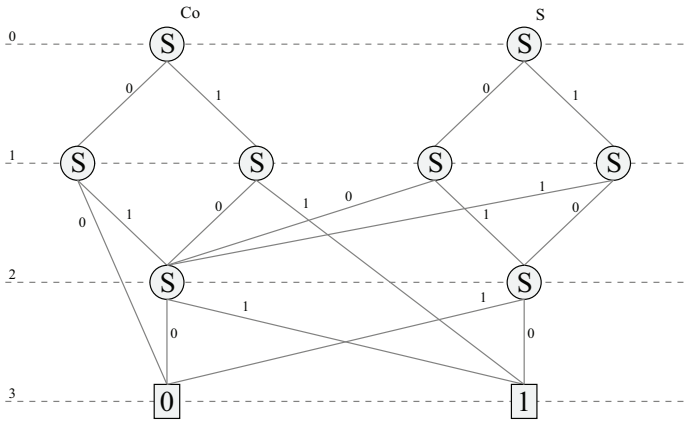


Figure 4.7: A shared decision diagram of the 2-variable full adder.

each obtained codeword and the length 6 vectors at distance ≤ 1 from the codewords. This procedure gives an error-correcting shared decision diagram for the full adder of 2 variables (Figure 4.8). The diagram has a terminal node labeled with $*$, since, due to the shortening, the used code is no longer perfect. Obtaining the value $*$ indicates two decision errors.

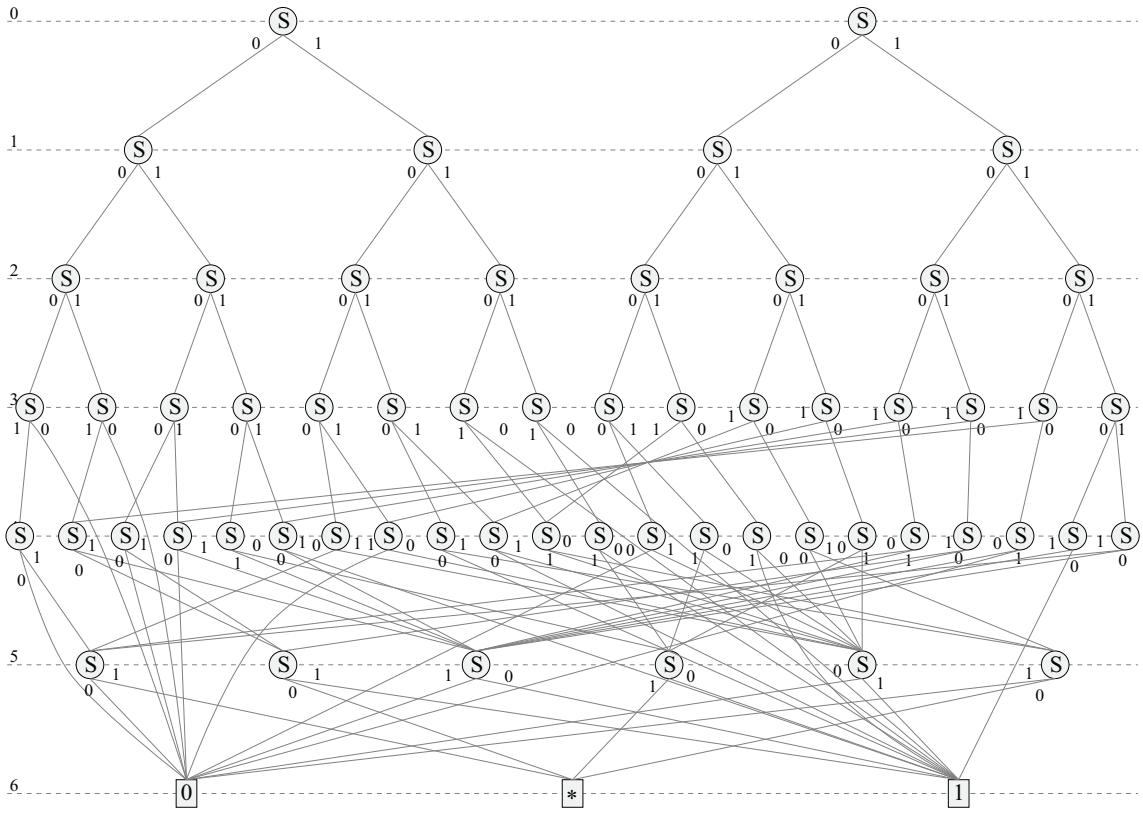


Figure 4.8: A shared robust diagram for the full adder of 2 variables using shortened Hamming code.

4.4.4 Ternary Hamming $(4, 2)$ Code

Hamming codes can be defined over any vector spaces $\mathbb{F}_q^n$. Non-binary Hamming codes exist with parameters $n = \frac{q^m-1}{q-1}$ and $k = \frac{q^m-1}{q-1} - m$. Choosing $q = 3$ and $m = 1$ gives the ternary $(4, 2)$ Hamming code, which is one-error-correcting. Using this code, we may construct a general error-correcting decision diagram for ternary functions of 2 variables.

The generator matrix $\mathbf{G}$ for the ternary $(4, 2)$ Hamming code is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 2 & 2 \end{bmatrix}.$$

The function g is obtained by multiplying $\mathbf{G}$ by the ternary vectors of length 2 and then mapping the obtained codewords and the length 4 ternary vectors within distance 1 from the codewords to the corresponding function values. The obtained ternary decision diagram will have 9 terminal nodes corresponding to $f(0,0), f(0,1), f(0,2), \ldots, f(2,2)$ (Figure 4.9). The code is perfect, hence there are no $*$-valued outputs in the obtained decision diagram.

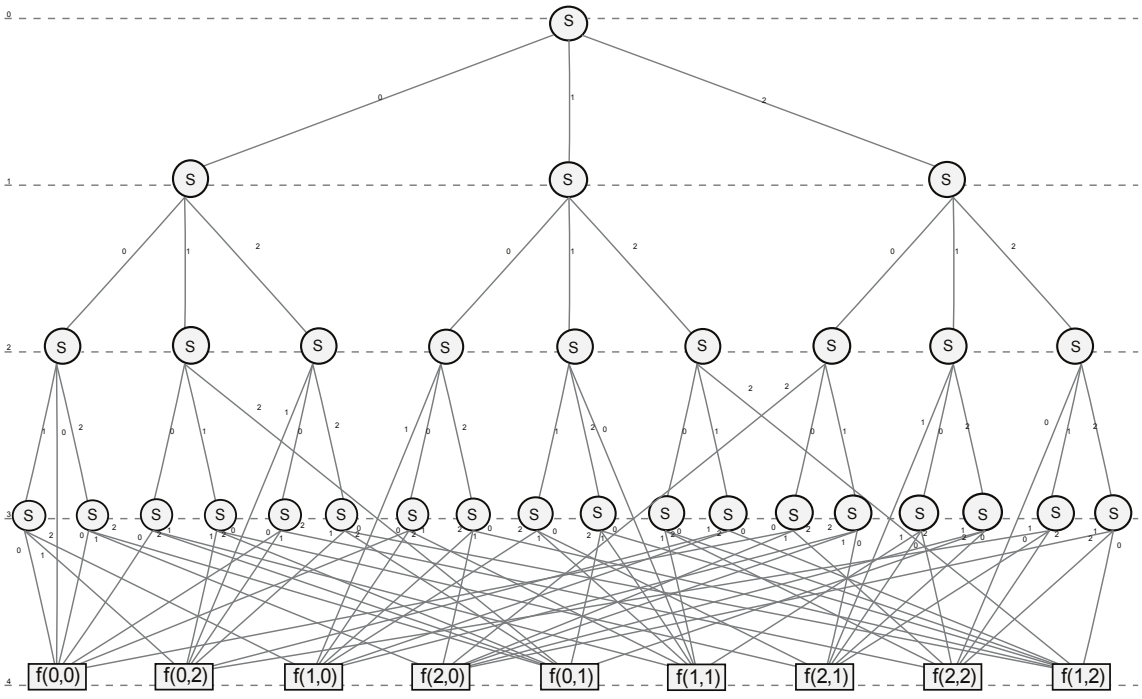


Figure 4.9: A general error-correcting decision diagram for ternary 2-variable functions using Hamming $(4, 2)$ code.

Similarly, as for binary functions, the robust decision diagram for a particular ternary function of 2 variables can be obtained by assigning the function values to the terminal nodes and reducing with respect to the function values.

For example, consider the ternary function defined as $f_9 = [1, 1, 2, 0, 1, 1, 2, 2, 0]^T$. If we assign the values of this function to the corresponding terminal nodes of the general diagram in Figure 4.9, we obtain the the robust diagram in Figure 4.10.

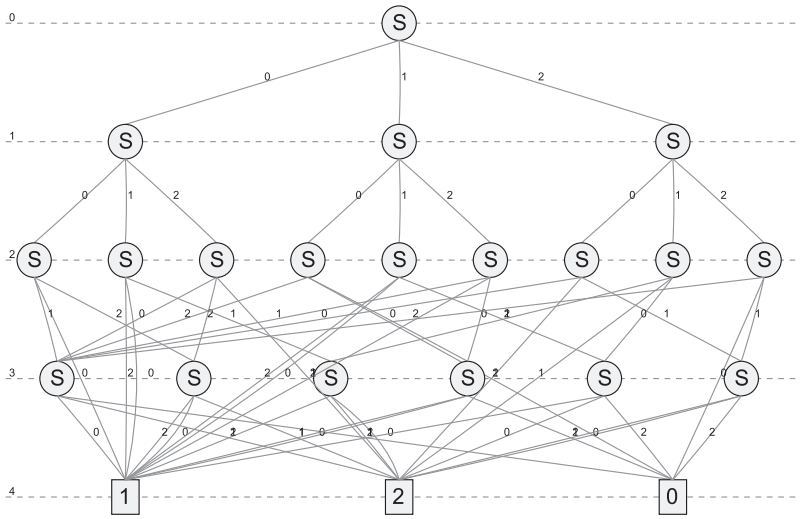


Figure 4.10: A robust diagram for f_9 using Hamming $(4, 2)$ code.

4.4.5 Repetition Codes

With binary repetition codes we can construct error-correcting decision diagrams, which have some analogy to the NMR technique described in Section 3.3. The resulting diagrams are redundant representations for single nodes.

The first example using such codes was already given in the introductory part of this chapter. In Figure 4.1, the error-correcting decision diagram generated with the $(3, 1)$ repetition code was shown. The diagram has no terminal nodes labeled as $*$, since every possible sequence is always decoded to either the value 0 or 1 by majority-vote decoding, i.e., every 3-bit sequence is always at a distance ≤ 1 from the sequence 000 or 111. It was discussed, how this example is in principle similar to the TMR technique. The key difference is that once we have reduced the decision diagram, the majority vote property is already included in the structure, i.e., no voters are needed in the circuit level implementation.

The binary $(5, 1)$ repetition code is two-error-correcting, and may also be used for generating a robust structure for a single node. With a larger code the robustness increases, but consequently the complexity of the BDD is higher (Figure 4.11).

Consider the $(3, 1)$ repetition code for $\mathbb{F}_q$. This code is perfect in $\mathbb{F}_2$, but will result in $*$-valued outputs for $\mathbb{F}_q$, where $q > 2$. However, we may use this code for generating a robust decision diagram for, e.g. quaternary (Figure 4.12) or 5-ary (Figure 4.13) logic. The resulting decision diagrams correct one decision error.

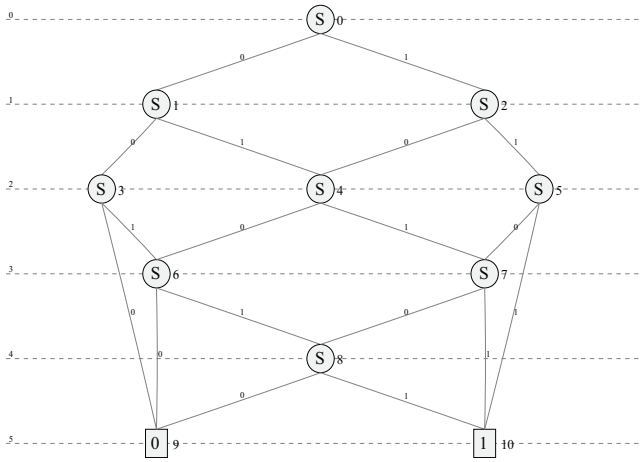


Figure 4.11: A robust BDD for a single binary node using the $(5, 1)$ repetition code.

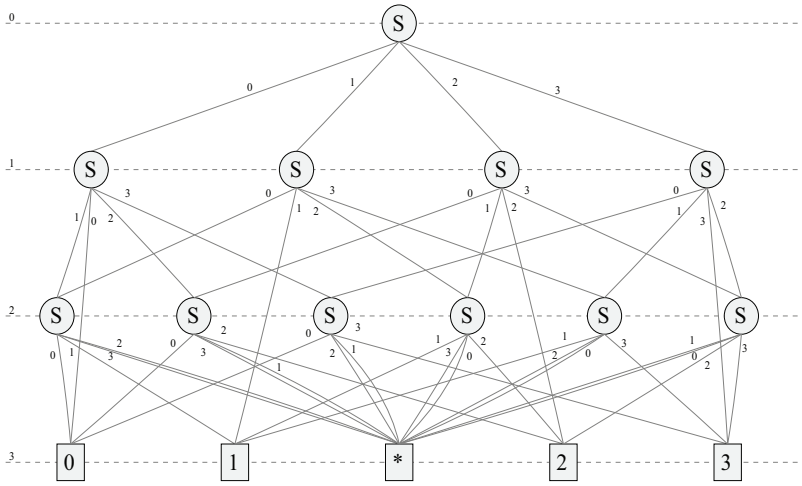


Figure 4.12: A robust diagram for a single quaternary node using the (3,1) repetition code.

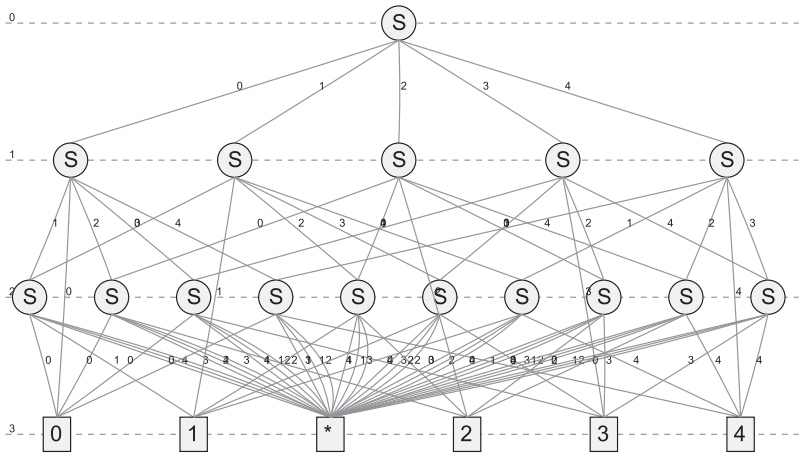


Figure 4.13: A robust diagram for a single 5-ary node using the (3,1) repetition code.

Notice that since the obtained error-correcting decision diagrams using repetition codes are robust representations of single nodes, we may replace the nodes of a traditional decision diagram of a function f with these robust diagrams to obtain a robust representation for f. This method of constructing a robust diagram is briefly discussed when analyzing the fault-tolerance of error-correcting decision diagrams in Section 5.3.

4.4.6 One-Lee-Error-Correcting Code for $q = 5$

The previous example for $q = 5$ is for representing a single decision node, but the number of non-terminal nodes in the robust diagram is much higher. However, we may construct a robust diagram using a perfect one-error-correcting code in the Lee metric, which will have significantly less non-terminal nodes.

Consider the one-Lee-error-correcting code having the generator matrix

$$\mathbf{G} = \begin{bmatrix} 3 & 1 \end{bmatrix}.$$

The construction of the robust diagram is done similarly as in the previous examples, but the mapping of length 2 vectors to corresponding values in $\mathbb{F}_5$ is done with respect to the Lee metric. Table 4.4 shows the radius one spheres around the codewords in the vector space $\mathbb{F}_5^2$, illustrating how the vectors $\mathbf{y}$ of length 2 satisfying $d_L(\mathbf{y}, \mathbf{xG}) \leq 1$, where $\mathbf{x} \in \mathbb{F}_5$, are found. The codewords, i.e., the vectors $[0,0]^T, [3,1]^T, [1,2]^T, [4,3]^T$ and $[2,4]^T$, are labeled by the corresponding $\mathbf{x} \in \mathbb{F}_5$ and in boldface, and the vectors of $\mathbb{F}_5^2$ at distance 1 from each codeword are labeled by the same $\mathbf{x} \in \mathbb{F}_5$ as the codeword.

Table 4.4: The radius one spheres around codewords (in boldface) labeled by the corresponding $\mathbf{x} \in \mathbb{F}_5$. The first component of $\mathbf{y}$ chooses the column and the second component chooses the row.

4	0	4	**4**	4	3
3	3	2	4	3	**3**
2	2	**2**	2	1	3
1	0	2	1	**1**	1
0	**0**	0	4	1	0
	0	1	2	3	4

The obtained robust diagram for a single 5-ary decision node is in Figure 4.14.

The increase in the number of non-terminal nodes is significantly less than with the $(3, 1)$ repetition code in the Hamming metric, and the obtained decision diagram has no terminal nodes labeled with the symbol $*$. However, the Lee metric introduces a functional difference to the diagram. In the error-correcting decision diagrams in

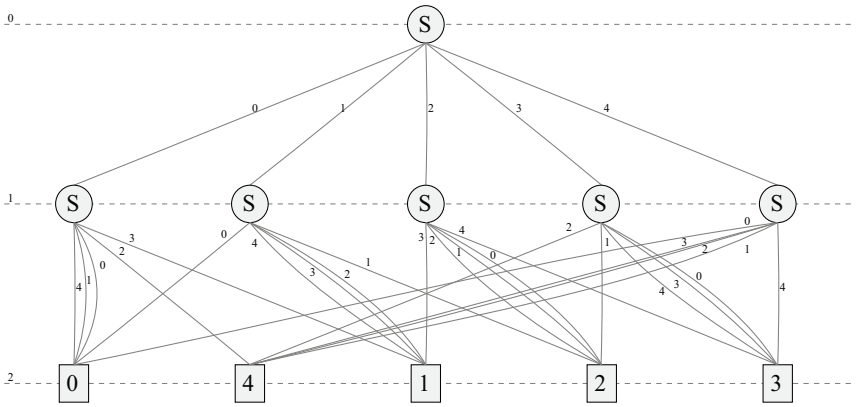


Figure 4.14: A robust diagram for a 5-ary node using a Lee-error-correcting code.

the Hamming metric, each line after a decision node is equal in the sense that whichever incorrect decision is made, the effect of the decision is the same in terms of error-correction. In the Lee metric, however, it is possible to make an error of value $\geq e$ in just one decision node, following that error correction is no longer possible.

5. FAULT-TOLERANCE ANALYSIS OF ERROR-CORRECTING DECISION DIAGRAMS

In addition to designing new fault-tolerance methods it is important to find ways to describe the performance and behavior of these systems with different design parameters. One approach to evaluating system reliability is to experimentally determine the reliability of a component as a function of time [2]. This requires testing several copies of components that can be too expensive or complex for such testing procedures. Therefore, it is necessary to describe the behavior of fault-tolerant systems such as error-correcting decision diagrams with reliability modeling techniques.

Reliability is defined by Naresky in [25] as "the ability of an item to perform a required function under stated conditions for a stated period of time". The reliability of a system can be described as a function of the reliability of a single module or component in the system. In this chapter, instead of modeling the reliability of error-correcting decision diagrams, we simply consider probabilities of correct and incorrect outputs with different combinations of nodes performing their functions correctly or incorrectly. Therefore, the effect of time is discarded, and the probability of correct or incorrect outputs with any input combinations can be computed as a function of the error probability of a single node in the diagram. The probability model should be considered already when designing the error-correcting decision diagrams. The selection of codes should be done with respect to the desired technology, since the fault model should fit the realization. For example, if it is assumed that the logical values can change into any incorrect values with equal probability, it is reasonable to use codes in the Hamming metric. When errors of different value have different probabilities, one can use, for example, codes in the Lee metric. Suitable codes can also be designed to fit the utilized technology best.

Modeling the probabilities of correct and incorrect outputs is discussed in Sections 5.1 and 5.2. Results of the fault-tolerance analysis for some example diagrams are given in Section 5.3. However, for larger diagrams, the exact determination of these probabilities can be difficult, and methods of approximation are discussed in Section 5.4.

5.1 The Probability Model for Diagrams based on Codes in the Hamming Metric

In this thesis, the performance of error-correcting decision diagrams is analyzed by determining the probability that the output is correct for any input in the robust diagrams. This probability is described as a function of the error probability of a single node in the diagram, i.e., the probability that the output of a node is incorrect. In the following, we consider error-correcting decision diagrams generated with codes in the Hamming metric. For concepts and notation of probability theory we use [12] as a reference.

We call decision nodes that make an incorrect decision faulty nodes. For traditional non-redundant decision diagrams, an incorrect output for the whole system is obtained whenever there is a faulty node on a path. For error-correcting decision diagrams, there may be up to e faulty nodes on a path, and the output is still correct. It could be so, that the correct output is obtained even if incorrect decisions are made in more than e nodes on a path, but we are not interested in these outputs.

We model the non-terminal nodes of an e-error-correcting decision diagram as independent binary random variables $\mu_1, \mu_2, \ldots, \mu_M$, where

$$\begin{aligned} &P\{\mu_i = 1\} = p \quad (1 \text{ means faulty}), \\ &P\{\mu_i = 0\} = 1 - p, \end{aligned}$$

for $i = 1, 2, \ldots, M$ and M is the total number of non-terminal nodes. Notice that when $q > 2$ the probability of a node not being faulty is still $(1 - p)$. This simplification is made, since in the Hamming metric, one incorrect decision is always at distance 1 from the correct value, due to the definition of the Hamming distance. Therefore, the effect on the total error is always the same no matter which incorrect output is given from a single node.

Now, denote by $P_1, P_2, \ldots, P_L$ the subsets of $\{1, 2, \ldots, M\}$ formed of the indexes of the non-terminal nodes in all paths from root to the terminal nodes. Using this notation, we can express the probability that the output of the error-correcting decision diagram is correct with any input as

$$\begin{aligned} &P\{\text{output correct for any input}\} \\ &= P\{\text{there are at most } e \text{ faulty nodes on any path}\} \\ &= P\{\max_{1 \le i \le L} \sum_{j \in P_i} \mu_j \le e\}. \end{aligned}$$

Since

$$\max_{1\leq i\leq L}\sum_{j\in P_i}\mu_j \leq \sum_{j=1}^{M}\mu_j, \tag{5.1}$$

we can write

$$\begin{aligned} P\{\text{output correct for any input}\} &\geq P\{\sum_{j=1}^{M}\mu_j \leq e\} \\ &= \sum_{i=0}^{e}\binom{M}{i}p^i(1-p)^{M-i}. \end{aligned}$$

Thus, we may write

$$P\{\text{output correct for any input}\} = \sum_{i=0}^{e}\binom{M}{i}p^i(1-p)^{M-i} + \sum_{i=e+1}^{M}\alpha_i p^i(1-p)^{M-i}, \tag{5.2}$$

where the coefficient α_i depends on the structure of the error-correcting decision diagram. Hence, for the probability of an incorrect output for the error-correcting decision diagram we have

$$\begin{aligned} P\{\text{incorrect output}\} &= 1 - \left(\sum_{i=0}^{e}\binom{M}{i}p^i(1-p)^{M-i} + \sum_{i=e+1}^{M}\alpha_i p^i(1-p)^{M-i}\right) \\ &= 1 - \left(1 - \sum_{i=e+1}^{M}\binom{M}{i}p^i(1-p)^{M-i} + \sum_{i=e+1}^{M}\alpha_i p^i(1-p)^{M-i}\right), \end{aligned}$$

which can be written in the form:

$$P\{\text{incorrect output}\} = \sum_{i=e+1}^{M}\left(\binom{M}{i} - \alpha_i\right)p^i(1-p)^{M-i}. \tag{5.3}$$

It is clear that in the expansion of the above equation, the lowest degree term is of degree at least $e+1$, i.e., of the form $A\cdot p^{e+1}$, where A is some constant. For a traditional diagram, since a single incorrect decision causes an incorrect output, the lowest degree term of the error probability function is always $B\cdot p$, where B is some constant.

5.2 The Probability Model for Diagrams based on Codes in the Lee Metric

When analyzing the fault-tolerance of robust decision diagrams constructed in the Lee metric, we have to take into account that in a decision node, we can make either the correct decision, an incorrect decision which is at Lee distance $\leq e$ from the correct value, or an incorrect decision which is at distance $> e$ from the correct value. For example, if $e = 3$, it is possible to make 3 incorrect decisions at distance 1, or an incorrect decision at distance 2 and an incorrect decision at distance 1, or one incorrect decision at distance 3, and still obtain the correct output. Therefore, the analysis must be slightly changed to make sense for the Lee-error-correcting decision diagrams.

The error-correcting decision diagrams based on codes the Lee metric are for q-ary logic where the variables are assumed to take value $0, 1, \ldots q-1$. In the following, for simplicity, we assume that $q = 2m+1$, i.e., that q is an odd integer of value ≥ 3.

It is natural to assume that larger errors are less likely than smaller errors. For example, depending on the technology, values within smaller distance from each other can be obtained with smaller difference in voltage in the circuit level, and it is therefore more probable that an incorrect value at a smaller distance is obtained. For instance, if $q = 5$, it is more likely for a 0 to change into the value 1 than into the value 2, i.e., the error $0 \rightarrow 1$ has a higher probability than the error $0 \rightarrow 2$.

We model the non-terminal nodes of a e-Lee-error-correcting decision diagram as independent random variables $\mu_1, \mu_2, \ldots, \mu_M$, and

$$P\{\mu_i = w\} = p_w,$$

where $w = 0, 1, 2, \ldots, m$, $0 < p_1 < p_2 < \cdots < p_m$, $p_0 = 1 - \sum_{w=1}^{m} p_w > 0$, and M is the total number of non-terminal nodes. Thus, $\mu_i = w$ is interpreted as a fault that causes a decision error of Lee-weight w. For example, let $q = 5$ and consider the node i. Then $\mu_i = 2$ is interpreted as decision errors $0 \rightarrow \{3, 2\}$, $1 \rightarrow \{4, 3\}$, $2 \rightarrow \{0, 4\}$, $3 \rightarrow \{1, 0\}$ and $4 \rightarrow \{2, 1\}$.

Similarly as for the Hamming metric, we denote by $P_1, P_2, \ldots, P_L$ the subsets of $\{1, 2, \ldots, M\}$ formed of the indexes of the non-terminal nodes in all paths from root to the terminal nodes.

Again, we may write

$$P\{\text{output correct for any input}\} = P\{\max_{1 \leq i \leq L} \sum_{j \in P_i} \mu_j \leq e\},$$

and due to equation (5.1), we can write

$$P\{\text{output correct for any input}\} \geq P\{\sum_{j=1}^{M} \mu_j \leq e\}.$$

It is difficult to write explicit formulas even for small values of e, but for given probabilities $p_0, p_1, \ldots, p_m$ and given n and e, we may compute $P\{\sum_{j=1}^{M} \mu_j \leq e\}$ as follows.

Expand the polynomial

$$\begin{aligned} A_M(x) &= (p_0 + p_1 x + p_2 x^2 + \cdots + p_m x^m)^M \\ &= p_0^M + A_1^{(M)} x + A_2^{(M)} x^2 + \cdots + A_e^{(M)} x^e + \cdots + p_m^M x^{mM}. \end{aligned}$$

Now, denote by $B_M(x) = p_0^M + A_1^{(M)} x + \cdots A_e^{(M)} x^e$. Then,

$$P\{\sum_{j=1}^{M} \mu_j \leq e\} = B_M(1).$$

This is because for a diagram having M non-terminal nodes, the coefficient $A_i^{(M)}$ is a sum of all such terms $p_{w_0} \cdots p_{w_s}$, where $w_0 \leq w_1 \leq \cdots \leq w_s \leq m$ and $w_0 + w_1 + \cdots + w_s = i$, therefore it corresponds to all combinations of faulty and not faulty nodes for which the total Lee-weight of the error is i.

For example, let $q = 5$, $e = 2$, $M = 6$, and $p_0 = 0.6, p_1 = 0.3, p_2 = 0.1$. Then,

$$A_6(x) = (p_0 + p_1 x + p_2 x^2)^6 = 0.047 + 0.140x + 0.222x^2 + \cdots,$$

and

$$B_6(x) = 0.047 + 0.140x + 0.222x^2.$$

Now we can evaluate $B_6(x)$ at 1 giving

$$P\{\sum_{j=1}^{6} \mu_j \leq e\} = B_6(1) = 0.409.$$

Notice that by writing $A_{M+1}(x) = A_M(x) A_1(x)$, we can obtain recursive formulas for the coefficients $A_i^{(M)}$.

If we simplify the model by assuming that $p_1 = p, p_2 = p^2, \ldots, p_w = p^w$, following

that $p_0 = 1 - \sum_{w=1}^{m} p^w$, then for the case $e \leq m$ we obtain the following formula:

$$\begin{aligned} &P\{\text{output correct for any input}\} \\ &= (1 - \sum_{w=1}^{m} p^w)^M + \sum_{s=1}^{e} \sum_{u=1}^{s} \binom{M}{u} \binom{s-1}{u-1} p^s (1 - \sum_{w=1}^{m} p^w)^{M-u} \\ &+ \sum_{s=e+1}^{M} \sum_{u=1}^{s} \alpha_u p^s (1 - \sum_{w=1}^{m} p^w)^{M-u}. \end{aligned} \tag{5.4}$$

The coefficient $\binom{M}{u}\binom{s-1}{u-1}$ is the number of ways to select u faulty nodes from the total M nodes of the diagram, which together produce an incorrect output at distance s from the correct outputs. This corresponds to first selecting any u nodes from the total M nodes, which is given by $\binom{M}{u}$, and then assigning the size of the error to each node in all possible orders, i.e., the number of ways of writing s as a sum of u positive integers, which is the u-composition of s given by $\binom{s-1}{u-1}$ [31]. The coefficient α_u depends on the structure of the diagram.

5.3 Results of the Fault-Tolerance Analysis

In this section, fault-tolerance analysis based on the probability models introduced in the previous sections is performed on some of the example decision diagrams of Section 4.4 to give some idea of their efficiency. The probability of a correct output with any input is computed as a function of p. The obtained probability is then compared to the probability of a correct output with any input of a non-redundant decision diagram performing the same function with no error correction. We compute the probabilities by a brute force method, i.e., by listing all the possible combinations of faulty and not faulty nodes, for which the output is correct. Since the coefficients of the higher order terms must be determined from the structure of the diagram, the example diagrams are not very large.

Consider the error-correcting BDD for a single node generated using the $(3, 1)$ repetition code (Figure 4.1). We want to list all the possible cases for which there is at most one faulty node on any path, following that the output of the diagram is always correct. The probability is given by equation (5.2), where the term α_i has to be determined separately from the diagram. Since there are 4 non-terminal nodes in the diagram, the first term is $(1-p)^4$ when there are no faulty nodes in the diagram, and the second term is $4p(1-p)^3$. The final term $p^2(1-p)^2$ is when there are exactly two faulty nodes in the diagram, situated after the root node on level 1 of the diagram. This gives the probability

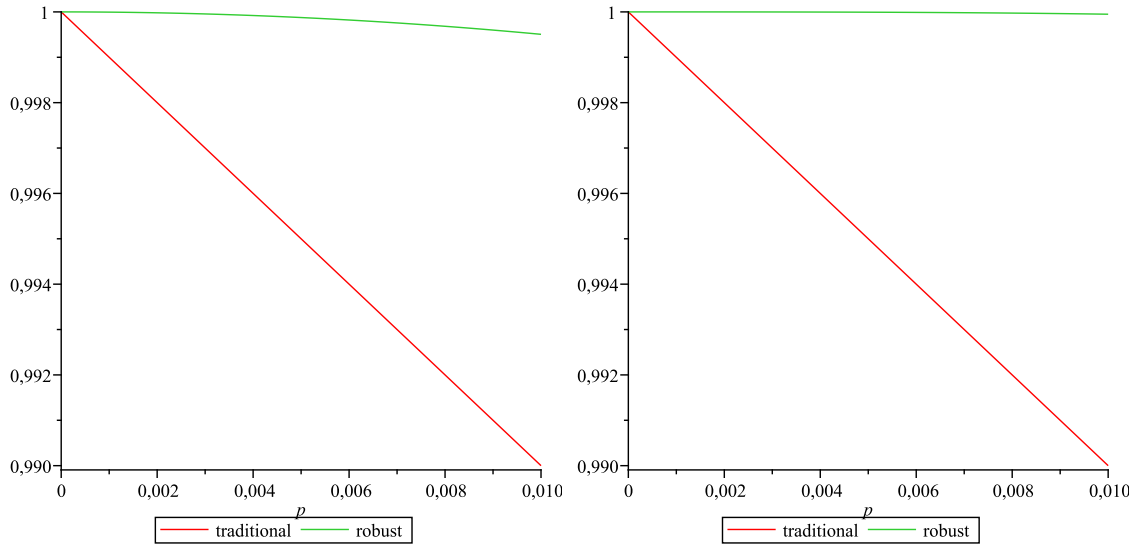
$$(1-p)^4 + 4p(1-p)^3 + p^2(1-p)^2$$

for a correct output with any input. Therefore, the probability that the output of the diagram is incorrect is

$$\tilde{p} = 1 - ((1-p)^4 + 4p(1-p)^3 + p^2(1-p)^2) = 5p^2 - 6p^3 + 2p^4. \tag{5.5}$$

The probability of a correct output $1 - \tilde{p}$ given any input is depicted together with the probability of a correct output of the single node decision diagram given by $1 - p$ in Figure 5.1a.

By similar computations, we obtain the probability of a correct output of the error-correcting BDD based on the $(5, 1)$ repetition code, which corrects two errors. This probability is shown together with the error probability of a single node in Figure 5.1b. Similarly, the probability of a correct output of the general error-correcting decision diagram of 2-variable functions in Figure 4.3 is depicted together with the traditional decision diagram for realizing 2-variable functions (Figure 5.2). Again, the probability that a node is faulty is p.



(a) (3,1) repetition code.

(b) (5,1) repetition code.

Figure 5.1: Comparison of the probability of a correct output of traditional and error-correcting binary decision diagrams generated using repetition codes.

Instead of mapping a function f into a robust one by some error-correcting code by equation (4.1), we may replace the nodes of the non-redundant decision diagram representing f by robust structures to obtain a robust representation for the given function f. It is also possible to replace just some of the nodes by robust structures, if it is reasonable to assume that in some parts of the circuit, errors are more likely than in others.

For example, consider a decision diagram for some binary function f, which has

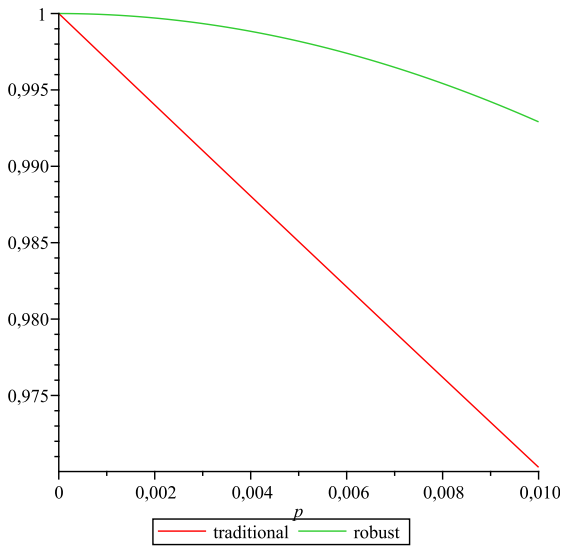


Figure 5.2: Comparison of the probability of a correct output of a traditional and the general error-correcting binary decision diagram generated using the (5,2) code.

M nodes in its reduced BDD. The error probability function for this diagram is $1-(1-p)^M$. Now, we may replace each node by the non-terminal nodes of the robust diagram generated using the $(3,1)$-repetition code. The resulting diagram will have $4M$ nodes in total and the error probability function $1-(1-\tilde{p})^M$, where $\tilde{p}$ is from equation (5.5). In Figure 5.3, the probability of a correct output of a diagram with $M = 50$ nodes is compared to the probability of a correct output of a diagram representing the same function, for which each node is replaced by the robust structure using $(3,1)$-repetition code.

Similarly as above for binary diagrams, we can determine the probability of a correct output of error-correcting decision diagrams of multiple-valued functions based on codes in the Hamming metric. For example, consider the general error-correcting decision diagram generated using the ternary Hamming $(4,2)$ code, which is shown in Figure 4.9. For this diagram $e = 1$ and there are in total $M = 31$ non-terminal nodes in the diagram. We need to list all the combinations of nodes for which there are either zero or one faulty nodes on each path.

Again, the first terms are directly given by equation (5.2) as $(1-p)^{31}$ and $31p(1-p)^{30}$. The following terms are determined by first considering all cases with exactly two faulty nodes in the diagram. It is clear that these faulty nodes can be situated on any single level of the diagram, since each node on a particular level never belongs to the same path as the other nodes on that level. It is also possible that the faulty nodes are on two different levels, but in this case we need to make sure that these faulty nodes are never on the same path. We may continue to list all possible cases when in the diagram there are exactly 3 faulty nodes, 4 faulty nodes, etc. The last case is when all the 18 nodes on level 3 are faulty. Adding up all these situations

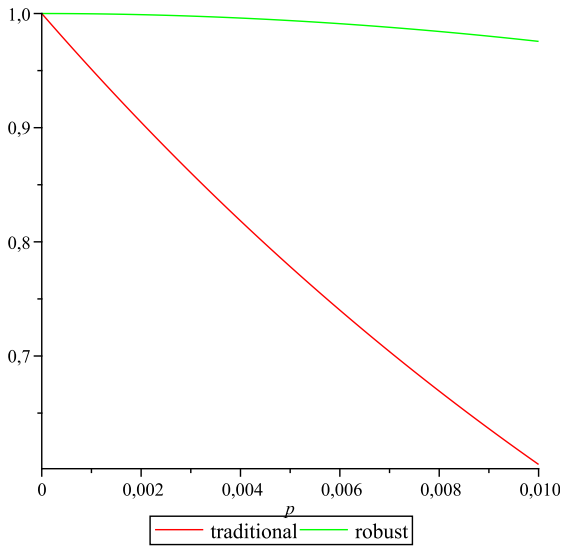
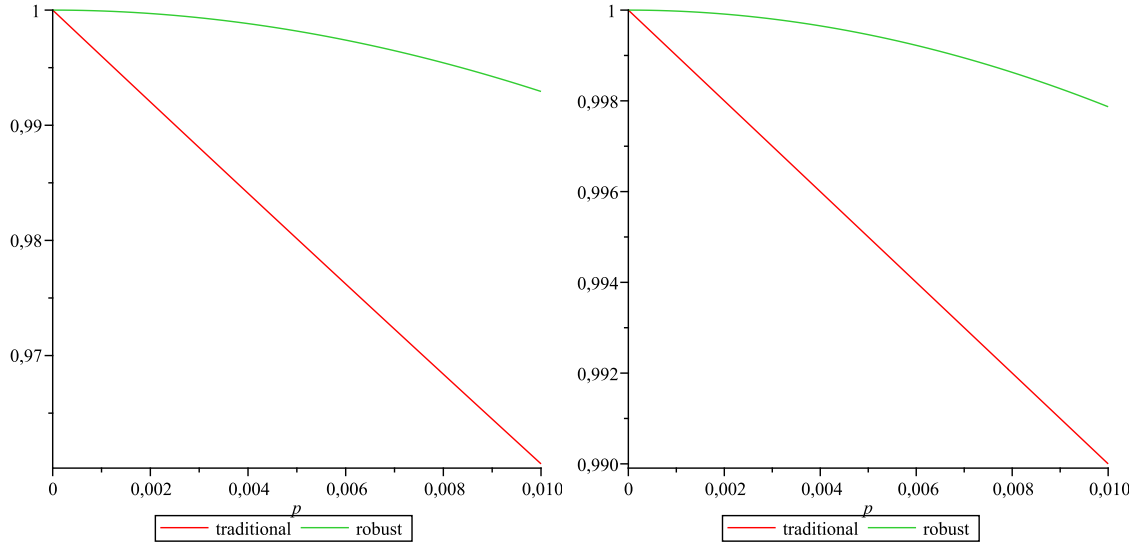


Figure 5.3: Comparison of the probability of a correct output of a traditional diagram with $M = 50$ nodes and a robust diagram, where each node is replaced by the robust structure generated using (3,1)-repetition code.

gives us the probability of a correct output with any input. We may now compare the obtained probability to $(1-p)^4$, which is the probability of a correct output in the corresponding traditional diagram (Figure 5.4a).

Consider now the error-correcting decision diagram in Figure 4.12, which is a robust construction corresponding to a single quaternary decision node. By our assumptions, a single quaternary node gives the correct output with probability $(1-p)$, since the probability of a faulty node is p. For the diagram in 4.12, we have $e = 1$ and in total 11 decision nodes, therefore the first terms of the probability function are given by $(1-p)^{11}$ and $11p(1-p)^{10}$. The rest of the terms are determined similarly as above for the ternary diagram. The obtained probability is depicted together with $(1-p)$ in Figure 5.4b.

Finally, take the diagram constructed using a one-Lee-error-correcting code in Figure 4.14, which has in total 6 nodes and $e = 1$. Since $q = 5$, an incorrect value can be at most at distance $m = 2$ from the correct value. Let us assume that the probabilities of decision errors are $p_1 = p$ and $p_2 = p^2$. Now, we may compute the probability for a correct output, starting with terms $(1-(p+p^2))^6$ and $6p(1-(p+p^2))^5$ given by equation (5.4). In addition to these, it is clear that the correct output is obtained even if all the nodes on level 1 give incorrect values at distance 1 from the correct value, if the top level node is correct. Therefore, the rest of the terms are given by $\sum_{i=2}^{5} \binom{5}{i} p^i (1-(p+p^2))^{6-i}$, when two or more of the five nodes on level 1 are faulty giving incorrect values at distance 1 from the correct value. The probability of a correct output of the robust diagram is compared to the probability of a correct output of a single 5-ary node in Figure 5.5.



(a) (4,2) Hamming code.

(b) (3,1) code for $q = 4$.

Figure 5.4: Comparison of the probability of a correct output of traditional and error-correcting multiple-valued decision diagrams based on codes in the Hamming metric.

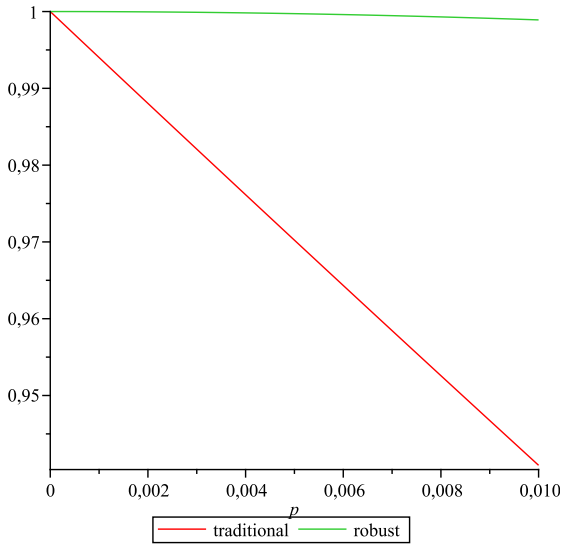


Figure 5.5: Comparison of the probability of a correct output of a traditional and a Lee-error-correcting decision diagram for $q = 5$.

5.4 Approximating the Probability of Correct Outputs

Computing the exact probability of correct outputs for error-correcting decision diagrams by brute force is very time-consuming, and nearly impossible for larger diagrams. Therefore, it is important to find ways to approximate the performance of these diagrams. The easiest approximation is given directly by the equation (5.1), following that we can obtain a lower bound for the probability of a correct output

with any input simply by computing $P\{\sum_{j=1}^{M} \mu_j \leq e\}$. It was shown in Sections 5.1 and 5.2 how this probability is computed in case of diagrams based on codes in the Hamming metric and Lee metric, respectively. To obtain a better estimate, the higher order terms should also be estimated in some way.

In the following, it is explained how the structure of the obtained diagrams can be used for estimating the higher order coefficients. The method is for diagrams based on codes in the Hamming metric, and it is based on the observation that the general decision diagrams generated using an (n, k) linear code always include the full tree structure on levels 0 to k. For a full tree structure, it is possible to derive a formula for determining the coefficients of the function describing the probability of correct outputs with any inputs.

An important property of linear codes is that each linear code is equivalent to a systematic code (Section 2.4). The systematic property induces a certain structure on general error-correcting decision diagrams, which are generated using linear codes. Since the generator matrix $\mathbf{G}$ of a systematic code includes the identity matrix $\mathbf{I}_k$, the codewords of the systematic code are such q^k length n vectors, which each begin with different length k vector. Now, the general error-correcting decision diagram is obtained by reducing the q-ary decision tree of the function g obtained with the mapping in equation (4.1). In any general error-correcting decision diagram, a path corresponding to a codeword must lead to a different terminal node. Therefore, the general error-correcting decision diagram generated using an (n, k) systematic code will have the full q-ary tree on levels 0 to k.

Since any linear (n, k) code C is equivalent to a systematic code, the systematic code is obtained from C by applying a fixed permutation of symbols to the codewords of C. Therefore, the q^k length n vectors, which each begin with different length k vector, must lead to different terminal nodes in the general error-correcting decision diagram of C. Thus, the general error-correcting decision diagram generated using the code C must similarly have the full q-ary tree on levels 0 to k.

Now, we may exploit this structure for estimating the probability of correct outputs of any diagram. For general diagrams, which are not reduced with respect to any particular functions, this approximation will give a lower bound for the probability of a correct output, and consequently an estimate for any particular function.

In this approach, the probability is approximated by estimating the higher order coefficients of the probability function. Consider the following for binary functions. To estimate the probability of a correct output of any robust BDD, we may form the first $e + 1$ terms by equation (5.2) and approximate the higher order coefficients by the coefficients of the probability of a correct output for the binary tree of depth k, for which there can be up to e faulty nodes on each path. Estimating the coefficients means that we approximate α_i of the term $\alpha_i p^i (1-p)^{M-i}$ (equation (5.2)) where M

is the number of nodes in the general robust BDD by S_i of the term $S_i p^i (1-p)^{2^k-1-i}$, where k is the depth of the binary tree and $M > 2^k - 1 \geq i$.

Thus, we may write for general error-correcting decision diagrams:

$$P\{\text{output correct for any input}\} \geq \sum_{i=0}^{e} \binom{M}{i} p^i (1-p)^{M-i} + \sum_{i=e+1}^{2^k-1} S_i p^i (1-p)^{M-i}$$

The above inequality holds, since for a given e-error-correcting decision diagram of M non-terminal nodes, the coefficient α_i equals the number of ways we can assign exactly i of the non-terminal nodes of the diagram faulty with at most e of them on any path. Similarly, for the depth k binary tree, which is strictly included in the diagram, the coefficient S_i equals the number of ways we can assign exactly i of the non-terminal nodes of the tree faulty with at most e of them on any path. It is clear that when assigning the faulty nodes to the diagram, we can start by assigning them to the binary tree part, obtaining the S_i ways, and after that consider the situations, where nodes that are not included in the tree can be faulty. Hence, $\alpha_i \geq S_i$.

Therefore, the approximation provides a lower bound for the probability of a correct output in case of general error-correcting decision diagrams. However, assigning a particular function may break the full binary tree structure and reduce the number of nodes in the diagram significantly. Thus, the approximation does not provide bounds for particular functions, but can be used as an estimate for the probability of a correct output for such diagrams.

The value of the coefficient S_i can be computed as follows. Let k be the depth of the tree, e the number of allowed faulty nodes on a path, and i the number of faulty nodes in the whole tree. Let $S_i = S(k, e, i)$ be the number of full depth k binary trees having i faulty nodes and up to e of them on any path. The number of such trees is equal to the coefficient of the term $p^i(1-p)^{2^k-1-i}$ in the probability of a correct output with any input for the full binary tree.

We may compute $S(k, e, i)$ for $k, e, i \geq 1$ recursively as follows:

$$\begin{aligned} S_i = S(k, e, i) &= \sum_{t=0}^{i} S(k-1, e, t) S(k-1, e, i-t) \\ &+ \sum_{t=0}^{i-1} S(k-1, e-1, t) S(k-1, e-1, i-1-t). \end{aligned} \tag{5.6}$$

The initial values are:

when $k = 0$: $S(k, e, i) = 1$ if $i \leq e$, where $i \leq 1$ and $S(k, e, i) = 0$ otherwise,
when $e = 0$: $S(k, e, i) = 1$ if $i = 0$ and $S(k, e, i) = 0$ otherwise,
when $i = 0$: $S(k, e, i) = 1$ always.

For example, the depth 0 tree consists of only the root node, hence $S(0, 0, 0) = 1$, since there is exactly one depth 0 tree having 0 faulty nodes, where on each path no faulty nodes are allowed.

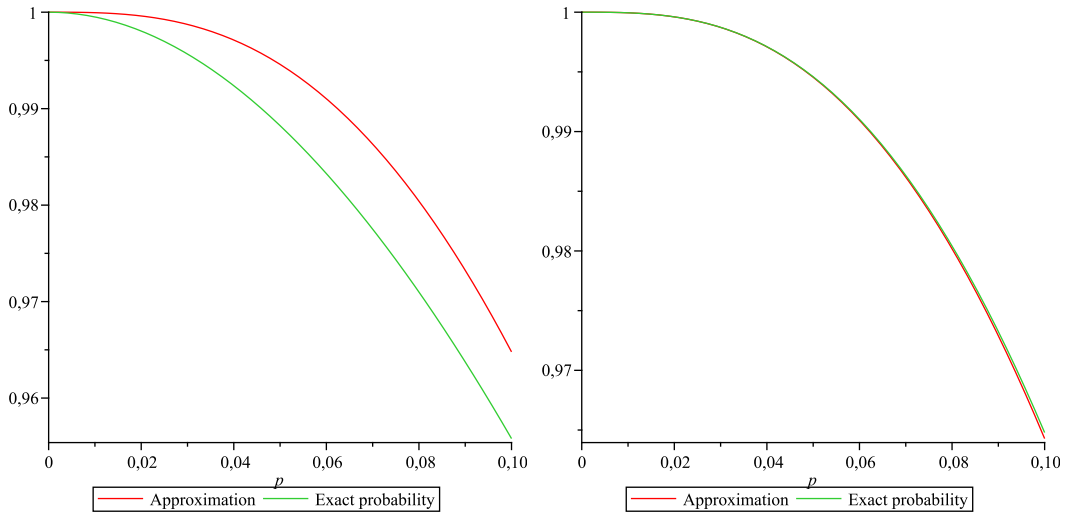
The formula associates two subtrees into a parent node, which may or may not be faulty. In the first convolution, the parent node is assumed to be non-faulty, and all of the exactly i faulty nodes must lie in the subtrees. The terms of the convolution consist of the different ways of assigning faulty nodes into the two subtrees. For example, in the very first term, all of the faulty nodes are in one of the subtrees. Consequently, in the last product of the convolution, all of the faulty nodes lie in the other subtree. In the second convolution, the parent node is assumed to be faulty, following that we must subtract 1 from e and i. The initial values guarantee that the recursion is well-defined.

An interesting observation in case of general BDDs is that the examples in Section 4.4 include not only the depth k binary tree structure, but in fact the full binary tree of depth $k + 1$. On the other hand, we might use for estimation such binary trees, which have approximately the same number of nodes as the obtained error-correcting decision diagram. For example, Figure 5.6 shows the comparison between the binary tree approximation and the exact probability of a correct output for robust representations of single nodes using $(3, 1)$ and $(5, 1)$ repetition codes. The binary tree used for the approximation is the depth $k + 1 = 2$ tree, where k is the number of variables of the original function. The binary tree of depth 2 is not included in the error-correcting decision diagrams, but the approximation still gives good results.

The above binary tree approximation can be generalized for the q-ary case. The general error-correcting decision diagrams of q-ary functions of k variables include the full q-ary tree of depth k. Therefore, the probability of a correct output with any input can be approximated using the coefficients of the probability function of this tree.

The equation (5.6) is generalized as follows.

$$\begin{aligned} S(k, e, i) = & \sum_{t_1 + \cdots + t_q = i} S(k-1, e, t_1) \cdots S(k-1, e, t_q) \\ + & \sum_{t_1 + \cdots + t_q = i-1} S(k-1, e-1, t_1) \cdots S(k-1, e-1, t_q). \end{aligned} \tag{5.7}$$



(a) (b)

Figure 5.6: Binary tree approximation compared to the exact probability of correct output for robust structure for a single node using $(3, 1)$ repetition code (a) and $(5, 1)$ repetition code (b).

The above method of approximation based on the structure of general error-correcting decision diagrams is one possible means of approximation, and it is particularly interesting because it is derived from the properties of linear codes. However, it is difficult to say anything about the efficiency of this approximation without extensive research. In case of very large functions, this estimate might not be much more efficient than simply using $P\{\sum_{j=1}^{M} \mu_j \leq e\}$ as a lower bound for the probability of a correct output. Other possible means of approximation would include, for instance, deriving bounds for the higher order coefficients based on the binomial expansion, since each coefficient α_i, when $i > e$, has to be less than the binomial coefficient $\binom{M}{i}$.

6. DISCUSSION

In this chapter, some remarks and explanations are given based on the theory and results presented in the previous chapters. The goal of the thesis was to describe a new method of designing fault-tolerant logic, which combines error-correcting codes with decision diagrams. In addition to this, the fault-tolerance of these systems was analyzed by deriving a probability model describing the probability of a correct output with any input. The example error-correcting decision diagrams were shown to have a significantly increased probability of a correct output than corresponding diagrams with no error correction.

6.1 Discussion on Error-Correcting Decision Diagrams

In Section 4.2 it was mentioned that the concept of error-correcting decision diagrams can be generalized to apply for nonlinear codes. The idea is simple, and can be described in terms of the encoding and decoding rules of the code. First, by the encoding rule of the code, the codewords of length n are determined and mapped to the corresponding $f(\mathbf{x}_i)$. Then, each non-codeword that is decoded to the information word $\mathbf{x}_i$ by the decoding rule of the code is mapped to $f(\mathbf{x}_i)$. Any other possible words of length n are mapped to the symbol $*$. Then, the decision diagram for the obtained function is generated, which is the error-correcting decision diagram of the function f. However, using linear codes has some advantages, since linear codes are known to have several good properties, and applying linear codes in the design of error-correcting decision diagrams is fast and straightforward in terms of generating the function g given by equation (4.1).

An important property of error-correcting decision diagrams is that depending on the technology, they are easy to implement, since the layout of the circuit is determined by the structure of the diagram. Therefore, the complexity of the circuit corresponds to the complexity of the robust diagram. Fault-tolerance is included already in the representations of functions, which means that no voters or checker circuits are needed in the designs. Since the diagrams lead to correct outputs with up to e faulty nodes on each path, a fairly large portion of the nodes can become temporarily or permanently faulty without affecting the output values.

The general error-correcting decision diagram of k-variable functions is basically a graphical representation of the decoding algorithm of the linear code, which is

used in construction of the diagram. The terminal nodes correspond to the q-ary length k vectors and each path corresponds to a received n-ary sequence. This way, each path leads to the k-ary sequence, which is interpreted from the received n-ary sequence in the decoding process. Therefore, the general error-correcting decision diagram of k-variable functions can be seen as a useful tool for the decoding procedure of the code. This leads to the idea of utilizing error-correcting decision diagrams when designing fast decoding circuits for error-correcting codes. When using error-correcting decision diagrams, the decoded sequence is directly available after the encoded sequence is received, once the structure of the error-correcting decision diagram is known. Moreover, since the structure of the error-correcting decision diagram of arbitrary functions contains the depth k binary tree, knowing the structure on levels $k+1$ to n would be sufficient. On the other hand, storing the decision diagram structure might not be efficient in case of very large codes, but the idea seems worth exploring. Also, representing the code by a decision diagram gives the idea of studying the properties of the code using the error-correcting decision diagram it generates.

6.2 Discussion on the Fault-Tolerance Analysis

The fault model in this thesis considers any faults that may cause an incorrect output in a node. Therefore, any type of fault that would cause the logical values to change inside the nodes to incorrect ones could lead to a faulty output of a node, which could result in the selection of an incorrect line. For example, in a stuck-at-fault the output of the gate is fixed, which can cause the data to propagate along the incorrect line after the faulty node.

The probability analysis done in Chapter 5 compares the performance of the robust constructions to the performance of non-redundant representations of functions. This analysis of robust decision diagrams shows, that the probability of incorrect outputs can be significantly decreased depending on the error-correcting properties of the code. With traditional diagrams, a single incorrect decision causes the output to be incorrect, and the lowest degree term of the error probability function is always a multiple of p, where p is the error probability of a single node in the diagram. For robust diagrams based on codes in the Hamming metric, the lowest degree term is always at least of degree $e+1$, i.e., of the form $A \cdot p^{e+1}$, where A is some constant. This means, that even with moderately high gate error probabilities, e.g, 10^{-2}, a robust construction will have a significantly decreased probability for an incorrect output. However, better error-correcting properties increase the complexity of the design.

The problem in analyzing the fault-tolerance of error-correcting decision diagrams is that due to their complex structure, in particular when large functions are consid-

ered, computing the exact probability is very time-consuming. Since each diagram has a different structure, the coefficients α_i and α_u in equations (5.2) and (5.4) have to be analyzed separately in case of each diagram, following that it is impossible to derive a universal formula for determining these coefficients. While computing the probabilities in Section 5.3, the possibility of using a script, which would go through each combination of faulty and not faulty nodes in a given error-correcting decision diagram with a given probability p of a decision error, and determine the overall probability of an incorrect output was considered, but this exhaustive method was found to be too heavy even for diagrams generated with the $(7, 4)$ Hamming code.

It was mentioned in the introductory part of Chapter 5 that the selection of the fault model depends on the utilized technology. By considering realizations using multiplexers, we can relate the well-known probability of correct decoding [8] to error-correcting decision diagrams. When implementing a switching function represented by a decision diagram using multiplexers, each level of multiplexers corresponding to one variable has a control line, which affects the output of each multiplexer on that level [6]. If we assume that the faults can only occur in the control lines, the probability of a correct output directly corresponds to the probability of correct decoding given by

$$P\{\text{correct decoding}\} = \sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i},$$

where n is the length of the codewords. This is because in the general error-correcting decision diagram constructed using a (n, k) linear code, there are in total n levels of non-terminal nodes, following that in the implementation with multiplexers, the circuit has n control lines. Therefore, there are always $\binom{n}{i}$ possible combinations of faulty control lines for each i. Clearly, the probability of a correct output is much higher with the above assumptions, than given by equation (5.2), since M, which is the total number of non-terminal nodes, is much larger than n. However, it is difficult to say wether this model would be applicable in reality.

To study the performance of error-correcting decision diagrams in more detail, it would be necessary to compare this method to other methods of increasing fault-tolerance. A correspondence between the TMR method and the robust diagram generated using the $(3, 1)$ repetition code is easy to notice. The simplest TMR construction (Figure 3.1) includes triplicated modules and a voter, which in comparison to the robust diagram in Figure 4.1 has similar complexity. The robust decision diagrams are constructed in such a way, that all nodes are equal in terms of weakness, whereas the voter in the TMR structure is unprotected from errors. Triplicating the voters provides more fault-tolerance, but consequently the complexity of the layout increases. Therefore, the robust decision diagram can be seen as a more efficient

structure. However, thorough comparison in terms of reliability would require more research and testing. Using the $(5, 1)$ repetition code provides protection against two errors, but the complexity is higher than of the TMR structure with triplicated voters. The comparison between more complex decision diagram structures and, for example, the TMR method is more difficult. It might not be straightforward to find efficient methods for comparing these.

7. CONCLUSIONS

In this thesis, fault-tolerance in digital systems was discussed, and a new method for providing fault-tolerance, combining error-correcting codes and decision diagrams, was presented. The method was discussed on a theoretical level, but due to the properties of decision diagrams, the implementation of the obtained robust structures, namely error-correcting decision diagrams, is straightforward depending on the desired technology. The ideas introduced in this thesis have also resulted in two conference papers [4], [5], where error-correcting decision diagrams and their performance is discussed.

In addition to easy implementation, a key advantage of error-correcting decision diagrams is that fault-tolerance is introduced already to the representations of functions, following that no additional checker circuitry is required in the obtained circuit layouts. The error-correcting decision diagrams in this thesis are based on linear codes, which are known to have several good properties and, in principle, easy implementation. Some of these properties, for example the systematic nature of the code, is visible in the obtained decision diagrams, and can be used when analyzing the performance of error-correcting decision diagrams.

The error-correcting properties of the diagrams depend on the properties of the code on which the diagram is based. In the fault-tolerance analysis of error-correcting decision diagrams, it was shown that even with moderately high gate error probabilities, e.g, 10^{-2}, an error-correcting decision diagram has a significantly decreased probability for an incorrect output in comparison with a non-redundant diagram. However, better error-correcting properties increase the complexity of the design, as more powerful codes are typically more complex.

Future work could include comparing error-correcting decision diagrams to other fault-tolerance methods, e.g. the TMR method, and determining the reliability of error-correcting decision diagrams by modeling and testing actual implementations. The possibility of studying the properties of error-correcting codes based on the obtained decision diagram is also worth exploring.

BIBLIOGRAPHY

[1] The webpage of The MathWorks, MATLAB. [www], [referenced 29.6.2011], Available: http://www.mathworks.com/products/matlab/.

[2] J.A. Abraham and D.P. Siewiorek. An algorithm for the accurate reliability evaluation of triple modular redundancy networks. *IEEE Transactions on Computers*, 23(7):682–692, 1974.

[3] S. B. Akers. Binary decision diagrams. *IEEE Transanctions on Computers*, 27(6):509–516, 1978.

[4] H. Astola, S. Stanković, and J. T. Astola. Error-correcting decision diagrams. In *Proceedings of The Third Workshop on Information Theoretic Methods in Science and Engineering*, Tampere, Finland, August 16-18, 2010.

[5] H. Astola, S. Stanković, and J. T. Astola. Error-correcting decision diagrams for multiple-valued functions. In *Proceedings of ISMVL 2011, 41th International Symposium on Multiple-Valued Logic*, pages 38–43, Tuusula, Finland, May 23-25, 2011.

[6] J. T. Astola and R. S. Stanković. *Fundamentals of Switching Theory and Logic Design.* Springer, 2006.

[7] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, 1968.

[8] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, 1983.

[9] R. E. Bryant. Graph-based algorithms for Boolean functions manipulation. *IEEE Transactions on Computers*, 35(8):667–691, 1986.

[10] S.K. Chilappagari and B. Vasic. Fault tolerant memories based on expander graphs. In *IEEE Information Theory Workshop, 2007. ITW '07*, pages 126 –131, 2007.

[11] M. M. Dickinson, J. B. Jackson, and G. C. Randa. Saturn V launch vehicle digital computer and data adapter. In *Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, AFIPS '64 (Fall, part I), pages 501–516, New York, NY, USA, 1964. ACM.

[12] William Feller. *An Introduction To Probability Theory And Its Applications*, volume 1. Wiley, 1971.

[13] J.B. Fraleigh. *A First Course in Abstract Algebra.* Addison-Wesley, 1974.

[14] K. Furutani, K. Arimoto, H. Miyamoto, T. Kobayashi, K. Yasuda, and K. Mashiko. A built-in Hamming code ECC circuit for DRAMs. *IEEE Journal of Solid-State Circuits*, 24(1):50 –56, February 1989.

[15] R. G. Gallager. *Low Density Parity Check Codes.* M.I.T. Press, 1963.

[16] S. Ghosh and P. D. Lincoln. Low-density parity check codes for error correction in nanoscale memory. SRI Computer Science Laboratory Technical Report, September 2007.

[17] W.J. Van Gils. A triple modular redundancy technique providing multiple-bit error protection without using extra redundancy. *IEEE Transactions on Computers*, 35(7):623–631, 1986.

[18] Thijs Krol. (N, K) concept fault tolerance. *IEEE Transactions on Computers*, 35(4):339–349, April 1986.

[19] Parag K. Lala. *An Introduction to Logic Circuit Testing.* Morgan & Claypool, 2009.

[20] Serge Lang. *Undergraduate Algebra.* Springer, New York, 2005.

[21] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, Amsterdam, 1997.

[22] J. Mathew, J. Singh, A.M. Jabir, M. Hosseinabady, and D.K. Pradhan. Fault tolerant bit parallel finite field multipliers using LDPC codes. In *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, pages 1684 –1687, May 2008.

[23] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations.* Morgan & Claypool, 2008.

[24] E. F. Moore and C. E. Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(3):191 – 208, 1956.

[25] Joseph J. Naresky. Reliability definitions. *IEEE Transactions on Reliability*, 19(4):198 –200, November 1970.

[26] Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, July 1990.

[27] D. K. Pradhan and J. J. Stiffler. Error-correcting codes and self-checking circuits. *Computer*, 13(3):27–37, March 1980.

[28] Tsutomu Sasao. *Memory-Based Logic Synthesis.* Springer, Heidelberg, 2011.

[29] Uwe Schöning. *Logic for Computer Scientists.* Birkhäuser, 2008.

[30] S. Stanković. XML-based framework for representation of decision diagrams. Ph.D. Thesis, Tampere University of Technology, Department of Signal Processing, 2009.

[31] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 2002.

[32] Heng Tang, Jun Xu, Yu Kou, S. Lin, and K. Abdel-Ghaffar. On algebraic construction of Gallager and circulant low-density parity-check codes. *IEEE Transactions on Information Theory*, 50(6):1269 – 1279, 2004.

[33] M. A. Thornton and V. S. S. Nair. Efficient calculation of spectral coefficients and their applications. *IEEE Transanctions on Computer-Aided Design of Integrated Circuits and Systems*, 14(11):1328–1341, 1995.

[34] J. H. van Lint. *Introduction to Coding Theory.* Springer Verlag, New York, 1982.

[35] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[36] J.F. Wakerly. Partially self-checking circuits and their use in performing logical operations. *IEEE Transactions on Computers*, 23(7):658 – 666, 1974.