Error Detection Codes for Arbitrary Errors Modeled by Error Graphs

Günther Nieß
University of Potsdam
niess@cs.uni-potsdam.de

Thomas Kern
Infineon Technologies AG
thomas.kern@infineon.com

Michael Gössel
University of Potsdam
mgoessel@cs.uni-potsdam.de

***Abstract*—In this paper arbitrary combinational errors are modeled by an error graph. The vertices of the error graph are n-tuples with q-ary components, and two vertices v and v' are connected by an edge from v to v' if the n-tuple v is changed by an combinational error into v'. If the error occurs with a known probability or frequency this can be modeled by weighted edges of the error graph. It is shown how an optimal error detecting code with $m \geq 1$ check bits and $k = n - m$ data bits can be determined from the error graph. For a small number of n an optimal solution can be computed. For larger numbers of n an efficient heuristic is proposed. As an example retention errors of multi-level NAND flash memory cells are modeled and the heuristic is used to determine efficient error detection codes. The new codes are compared with error detection codes for unidirectional errors with a limited magnitude by experimental results.**

I. Introduction

In this paper it is shown how $m \geq 1$ optimal check bits of an error detecting code can be determined for an arbitrary given error model. Errors are modeled by an error graph where vertices of the error graph are n-ary binary words. Two vertices v_1 and v_2 are connected by an edge from v_1 to v_2 if an error of the error model changes the bits of v_1 into the bits of v_2. This graph based error model allows adequately to express all combinational errors, caused by different fault mechanisms. It is shown how probabilities of the corresponding errors can be assigned to the edges as weights.

$m \geq 1$ check bits for a given error model are determined by an algorithm based on graph theory minimizing the number of edges of the error graph. Since the edges correspond to different errors of the error model the number of remaining errors after determining the optimal check bits is minimal. It is also shown how additional check bits can be added to the check bits of an already defined code.

The paper is a generalization of [8] with respect to the error graph and the algorithm for the determination of check bits. The error graph in [8] has only unweighted edges and the corresponding algorithm determines a single check bit at a time. This paper describes error graphs with error probabilities and the new heuristic determines $m \geq 1$ check bits of the error detection code C in one step for the error graph with weighted edges.

The paper is organized as follows. In section II the error model is introduced and it is shown how different types of faults can be expressed by the error graph. Section III demonstrates how the error model is used to define an error detection code or to assign m additional check bits to a given code. To be able to handle also larger numbers of data bits a heuristic for the determination of additional check bits is described. The application of the error graph and the heuristic to assign additional check bits is presented in section V for multi-level flash memory cells and retention errors. Section VI concludes the paper.

II. Error Model

In this section the error model used in this paper is introduced. The error model is described by an error graph. The vertices of the error graph are n-ary words. If an error exists altering a word v into a word v' the vertex v is connected by a directed edge with the vertex v' from v to v'. If there exists also an error changing v' into v the vertices v and v' are connected by an undirected edge. If the error appears with the probability p, the edge from v to v' has the weight p. Under the assumption that for every error e which changes the word v to v' an error exists which transforms v' in v, the graph is undirectional.

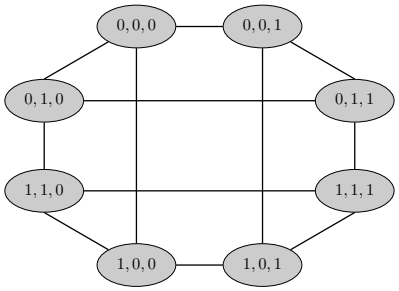

Fig. 1. Error graph for single bit errors of binary words with length 3

In Fig. 1 an error graph for single bit errors of binary data words of length 3 is shown. The vertices of the error graph can represent for example the state of 3 binary memory cells or data transmitted over a bus. A single bit error flips the value of one bit into its opposite value. For example an error modifying the last bit of the stored word $(1, 0, 1)$ into $(1, 0, 0)$ is expressed in Fig. 1 by the edge connecting the vertices $(1, 0, 1)$ and $(1, 0, 0)$.

In electrical circuits faults occur changing the correct behavior of the circuit only if some conditions are fulfilled. The single stuck-at fault model is the most frequently used for integrated circuits. For example a stuck-at-0 fault on a line

only changes the behavior if the line should carry a 1. It is well-known that a single stuck-at fault only leads in half of the possible configurations to a functional error. But if there are two stuck-at faults then in only 25% of the possible inputs two lines will be erroneous. In 50% of the inputs will occur a single bit error and in 25% the fault will not lead to a functional error.

Stuck-at faults can be easily modeled by an error graph as explained in Fig. 2. We consider 4 bits u_1, u_2, u_3, u_4. The error graph has the 16 vertices $(0,0,0,0)$, $(0,0,0,1)$, $\ldots$, $(1,1,1,1)$. If we assume that u_1 and u_2 are stuck-at-0 we obtain the error graph of Fig. 2. The vertices $(1,1,0,0)$, $(1,0,0,0)$, $(0,1,0,0)$ are erroneously changed into the word $(0,0,0,0)$. Also the data words $(1,1,0,1)$, $(1,0,0,1)$, $(0,1,0,1)$ are changed into $(0,0,0,1)$, the words $(1,1,1,0)$, $(1,0,1,0)$, $(0,1,1,0)$ are changed into $(0,0,1,0)$ and the words $(1,1,1,1)$, $(1,0,1,1)$, $(0,1,1,1)$ are changed into $(0,0,1,1)$. Therefore in the error graph shown in Fig. 2 the vertices $(1,1,0,0)$, $(1,0,0,0)$, $(0,1,0,0)$ are connected by an edge to the vertex $(0,0,0,0)$, and also each error listed above is represented in the error graph with a directed edge.

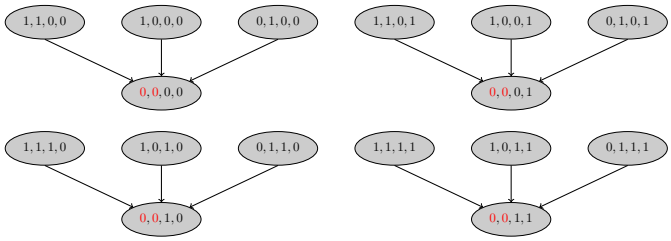

Fig. 2. Error graph for two stuck-at-0 faults

Many codes have been published to correct and detect asymmetric and unidirectional errors with a limited magnitude [3], [4], [6], [7], [5]. The most common application for these codes are error correction and detection for multi-level flash memory cells [3], [7]. Such a memory cell has multiple threshold voltage levels for different programmed values. A unidirectional error with a limited magnitude changes the programmed values of memory cells by only a limited absolute value in one direction, increasing or decreasing. An asymmetric error with a limited magnitude exclusively increases or only decreases the programmed value by a limited value.

TABLE I
MULTI-LEVEL FLASH CELL THRESHOLD VOLTAGE DISTRIBUTION

Stored electrons	lowest amount			highest amount
Binary encoding	11	10	01	00
Gray encoding	10	11	01	00

In Fig. 3 an error graph for 2 multi-level memory cells and asymmetric errors of magnitude $l = 1$ is shown. Each memory cell can be programmed with 2 bits and can therefore store the values 0 (00), 1 (01), 2 (10) and 3 (11). The cell state is defined by the amount of electrons on a storage layer. There are mainly two methods used to interpret the charge programmed into a multi-level flash cell shown in Table I. The first method uses the Gray encoding for interpreting the amount of electrons of a flash cell. The Gray code differ for two successive programmed levels of a flash cell in only one bit. Using this method ensures that a small change in the programmed charge of a flash cell results in only a small number of bit errors of the stored value. The second method called binary encoding in Table I counts the level of a multi-level flash cell starting with the highest amount of electrons as 0. If the cell looses a small amount of electrons over time using the second method then the represented value changes by a small magnitude as well. For example the two cells were programmed to store the values $(0,0)$. If a limited magnitude error occur the content of the cells may be interpreted as $(0,1)$, $(1,0)$ or $(1,1)$. These three errors are represented in the error graph with the three edges connecting the vertex $(0,0)$ to the vertices $(0,1)$, $(1,0)$ and $(1,1)$.

Detailed analysis for NAND flash memory chips revealed that the retention errors depend on the values programmed into the flash cells [2]. Retention errors are errors when the value stored in a flash cell changes over time. Usually this happens when the charge programmed in the floating gate disappear gradually through leakage current.

For the considered chips the probability of retention errors with disabled error correction (ECC) was after 3 years and 3,000 programming and erasure cycles approximately $q \approx 10^{-4}$ and the most common retention errors were $00 \to 01$, $01 \to 10$, $01 \to 11$ and $10 \to 11$ with their relative percentage over all retention errors 46%, 44%, 5% and 2%. If we assume the error rate of the flash memory cells is statistically independent and the programmed values of the memory cells are equally distributed we can calculate the probability of the erroneous transitions of the memory cells. The remaining three percent of the retention errors are not further specified in the paper [2] and are therefore not modeled.

Fig. 4 shows an error graph for 2 flash memory cells with the modeled retention errors from [2]. The error distribution is modeled with weighted edges where the weights corresponds to the error probability.

If we compare the asymmetric error model with limited magnitude from Fig. 3 with the error graph from Fig. 4 which is derived from experimental results [2] we can see the similarity. But if we compare the graphs in more detail, we can discover that in the error graph in Fig. 4 not only additional errors e.g. between the vertices $1,1$ $(01,01)$ and $1,3$ $(01,11)$ are modeled they are also much more likely then some errors from the error graph of Fig. 3, e.g. the error between the vertices $2,2$ $(10,10)$ and $3,3$ $(11,11)$.

III. DETERMINATION OF ADDITIONAL CHECK BITS

In this section we describe how additional check bits for error detecting block codes can be determined. We consider two cases.

1) In the first case we determine m optimal check bits c_1, c_2, $\ldots$, c_m for k data bits u_1, u_2, $\ldots$, u_k. Thereby we

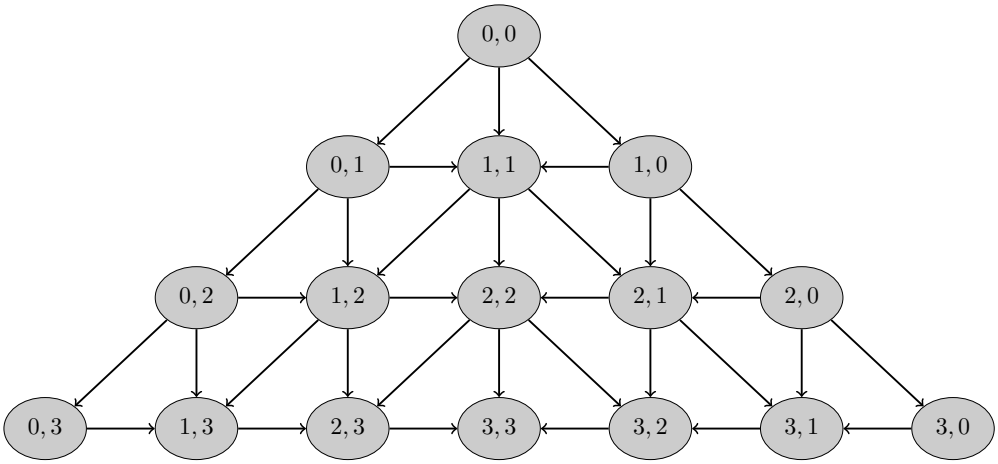

Fig. 3. Error graph for multi-level memory with asymmetric errors of limited magnitude

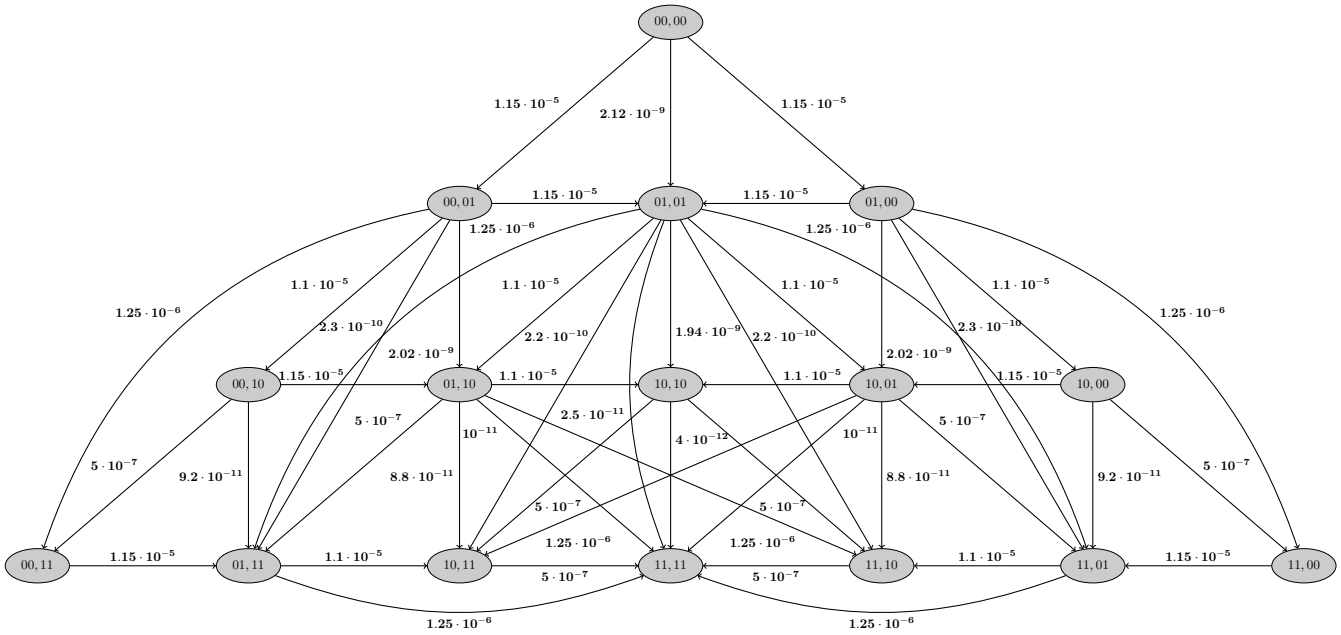

Fig. 4. Error graph for multi-level flash memory with retention error distribution

assume that there are 2^k different k-tupels of data bits. When the check bits c_1, c_2, ..., c_m are determined by the k-ary Boolean functions f_1, f_2, ..., f_m from the data bits u_1, u_2, ..., u_k as $c_1 = f_1(u_1, u_2, \ldots, u_k)$, $c_2 = f_2(u_1, u_2, \ldots, u_k)$, ..., $c_m = f_m(u_1, u_2, \ldots, u_k)$ the data bits and the corresponding check bits u_1, u_2, ..., u_k, c_1, c_2, ..., c_m are forming a code word of a code C. The code C is optimal, if the number of undetected errors or if the sum of error probabilities of the undetected errors is minimal.

2) In the second case we assume that the bits (v_1, v_2, ..., $v_{n'}$) are to be protected by additional check bits. We assume the bits (v_1, v_2, ..., $v_{n'}$) are already code words of an error detecting code C_{out}. In this case we add optimal additional bits c_1, c_2, ..., c_m to improve the error detection capability of the code C_{out}. The check bits are determined as $c_1 = g_1(v_1, v_2, \ldots, v_{n'})$, $c_2 = g_2(v_1, v_2, \ldots, v_{n'})$, ..., $c_m = g_m(v_1, v_2, \ldots, v_{n'})$ for (v_1, v_2, ..., $v_{n'}$) $\in C_{out}$. The code C_{out} can be considered as an outer code and when the check bits c_1, c_2, ..., c_m are determined by the Boolean functions, the bits (v_1, v_2, ..., $v_{n'}$, c_1, c_2, ..., c_m) form a codeword of the determined inner code C_{in}.

Now we consider the first case and assume that a initial error graph with all 2^n possible n-ary binary vertices is given. So errors can be modeled in the data part and the check bits.

For a given data word $u = (u_1, u_2, \ldots, u_k)$ we denote the set of all vertices with the same data bits and different check

bits as equal data bit nodes

$$ED(u_1, u_2, \ldots, u_k) := \{(v_1, v_2, \ldots, v_{k+m}) \in V \mid u_1 = v_1, \\ u_2 = v_2, \ \ldots, \ u_k = v_k\}$$

where V is the set of vertices of the error graph. An initial error graph $ED(u)$ consists of 2^m words with 2^m different values for the m check bits. For every vertex $v = (u, c) = (u_1, u_2, \ldots, u_k, c_1, c_2, \ldots, c_m)$ the set of complementary vertices $CV(v)$ is defined as

$$CV(u, c) := ED(u) \setminus \{(u, c)\}.$$

For an initial error graph the set of complementary vertices of a vertex v consists of all $2^m - 1$ vertices with the same data bits as v but with different check bits.

To determine an error detection code C for each data word u a vertex $v \in ED(u)$ has to be selected as codeword $v \in C$ and all complementary vertices $v' \in CV(v)$ have to be removed from the error graph. For example if the vertex $(u_1, u_2, \ldots, u_k, 1, 0, \ldots, 0)$ is selected as codeword from the error graph, we have the check bit function $1 = f_1(u_1, u_2, \ldots, u_k)$, $0 = f_2(u_1, u_2, \ldots, u_k)$, $\ldots$, $0 = f_m(u_1, u_2, \ldots, u_k)$. To determine the Boolean functions $f_1, f_2, \ldots, f_m$ one of the vertices for every data word has to be selected as codeword.

This can be done in $(2^m)^{2^k}$ different ways. For small values of k and m, $m \cdot 2^k \leq 64$, it is possible to consider all cases to determine the best code. For larger values of k and m a heuristic approach was developed. At every step of the algorithm a vertex is selected as a codeword and $2^m - 1$ vertices are removed from the error graph.

For every vertex $v = (u, c) \in V$ we assign a weight $weight(v)$ which is determined as the sum of the weights of the edges connecting v from or to all vertices of the actual error graph which are not elements of the equal data bit nodes $ED(u)$. If the graph is unweighted we count each edge with weight 1. If a vertex $v = (u, c) \in ED(u)$ is selected as a codeword of the code C then its edges connecting v with vertices outside $ED(u)$ remain unchanged. But all the complementary vertices $v' \in CV(u)$ are removed together with all their edges from the actual error graph. To select the best vertex as a codeword for every vertex $v \in V$ we introduce the rank of vertex as

$$rank(v) := \left(\sum_{v' \in CV(v)} weight(v') \right) - weight(v).$$

The sum over the complementary vertices $v' \in CV(v)$ describes the reduction of weights by removing all the vertices from $CV(v)$ and $rank(v)$ takes the remaining edges from the vertex v into account. If the edges represent the error probabilities, then the $rank$ of a vertex v is the additional error detection probability which is gained by selecting v as a codeword, minus the probability of additional undetected errors.

The heuristic selects in every step a vertex v with a maximal rank as codeword and removes the complementary vertices $CV(v)$. So in every step a new codeword of the code C is determined and after 2^k steps the code C is completely determined.

For example in Fig. 4 the equal data nodes of the data word 01 are $ED(01) = \{(01, 00), (01, 01), (01, 10), (01, 11)\}$, the complementary vertices of the vertex $(01, 01)$ are $CV(01, 01) = \{(01, 00), (01, 10), (01, 11)\}$, the $weight$ of the vertex $(01, 01)$ is calculated by

$$\begin{aligned} weight(01, 01) &= 2.12 \cdot 10^{-9} + 1.25 \cdot 10^{-6} + 1.1 \cdot 10^{-5} \\ &\quad + 2.2 \cdot 10^{-10} + 1.94 \cdot 10^{-9} + 2.5 \cdot 10^{-11} \\ &\quad + 2.2 \cdot 10^{-10} + 1.15 \cdot 10^{-5} \\ &= 2.3754525 \cdot 10^{-5} \end{aligned}$$

and the rank of this vertex is

$$\begin{aligned} rank(01, 01) &= \left(\sum_{v \in \{(01,00),(01,10),(01,11)\}} weight(v) \right) \\ &\quad - weight(01, 01) \\ &= 2.375225 \cdot 10^{-5} + 2.3752118 \cdot 10^{-5} \\ &\quad + 2.3750322 \cdot 10^{-5} - 2.3754525 \cdot 10^{-5} \\ &= 4.7500165 \cdot 10^{-5}. \end{aligned}$$

All ranks of the error graph of Fig. 4 are listed in Table II.

TABLE II
RANKS OF THE INITIAL ERROR GRAPH OF FIG. 4

vertex v	$(00, 00)$	$(00, 01)$	$(00, 10)$	$(00, 11)$
$rank(v)$	$2.30002 \cdot 10^{-5}$	$2.30000 \cdot 10^{-5}$	$2.30043 \cdot 10^{-5}$	$2.30045 \cdot 10^{-5}$
vertex v	$(01, 00)$	$(01, 01)$	$(01, 10)$	$(01, 11)$
$rank(v)$	$4.75047 \cdot 10^{-5}$	$4.75002 \cdot 10^{-5}$	$4.75050 \cdot 10^{-5}$	$4.75086 \cdot 10^{-5}$
vertex v	$(10, 00)$	$(10, 01)$	$(10, 10)$	$(10, 11)$
$rank(v)$	$2.30043 \cdot 10^{-5}$	$2.30002 \cdot 10^{-5}$	$2.30006 \cdot 10^{-5}$	$2.30038 \cdot 10^{-5}$
vertex v	$(11, 00)$	$(11, 01)$	$(11, 10)$	$(11, 11)$
$rank(v)$	$3.50068 \cdot 10^{-6}$	$3.50004 \cdot 10^{-6}$	$3.50006 \cdot 10^{-6}$	$3.50058 \cdot 10^{-6}$

Fig. 5 illustrates the process of assigning additional check bits with the proposed heuristic. First the algorithm selects a set of vertices $ED(u^*)$ where the vertex $v^* = (u^*, c^*)$ has the highest rank. After removing all edges which are connected to or from one vertex v' of the complementary vertices $v' \in CV(v^*)$ the vertices $CV(v^*)$ can be removed from the graph. If the check bit functions aren't completely defined the algorithm continuous with the next vertex for removal. The algorithm repeats the above steps until the code is defined.

Following this algorithm according to Table II the vertex $v = (01, 11)$ has the maximum rank and is therefore selected as codeword and the complementary vertices $CV(01, 11) = \{(01, 00), (01, 01), (01, 10)\}$ as well as every edge connected to one of the complementary vertices $CV(01, 11)$ are removed from the error graph. The resulting error graph is shown in Fig. 6 where the new ranks are listed above and below the vertices. The vertex $v = (10, 10)$ has now the highest rank and is therefore selected as codeword. Now the algorithm remove all edges connected to or from one of the complementary vertices $CV(10, 10) = \{(10, 00), (10, 01), (10, 11)\}$ and then remove the complementary vertices $CV(10, 10)$ itself from the error graph. In the resulting error graph, the vertex $(00, 00)$ has the

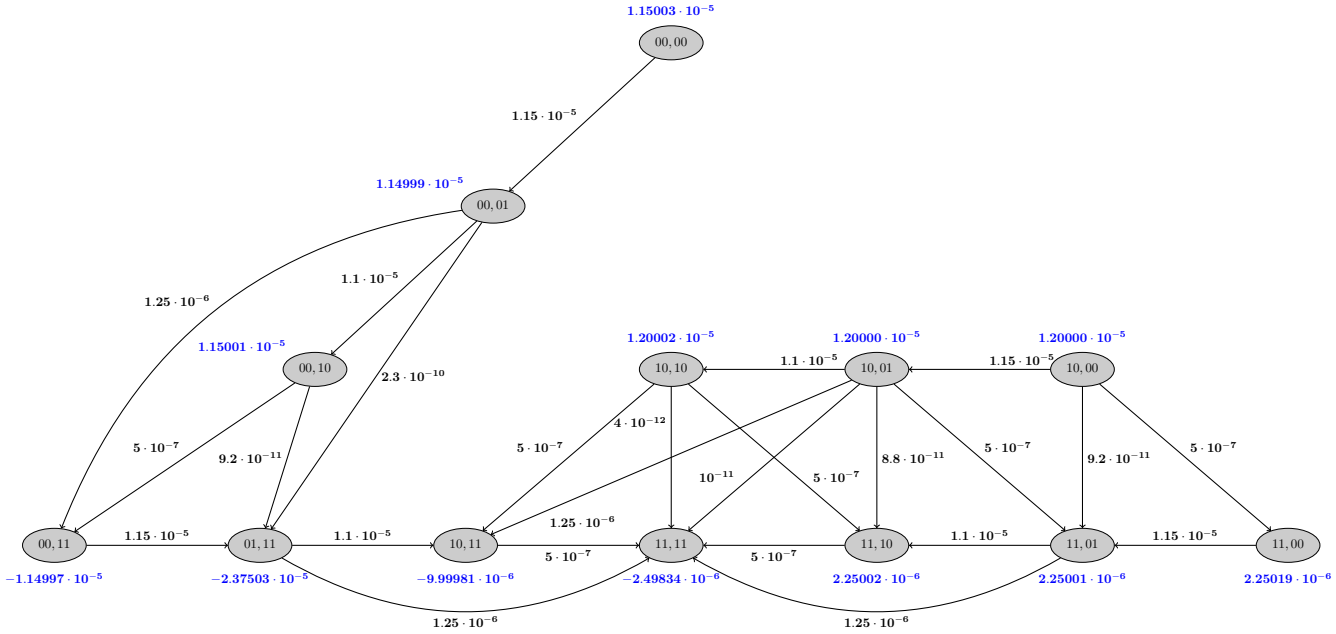

Fig. 6. Intermediate weighted error graph after selecting vertex $(01, 11)$ as codeword

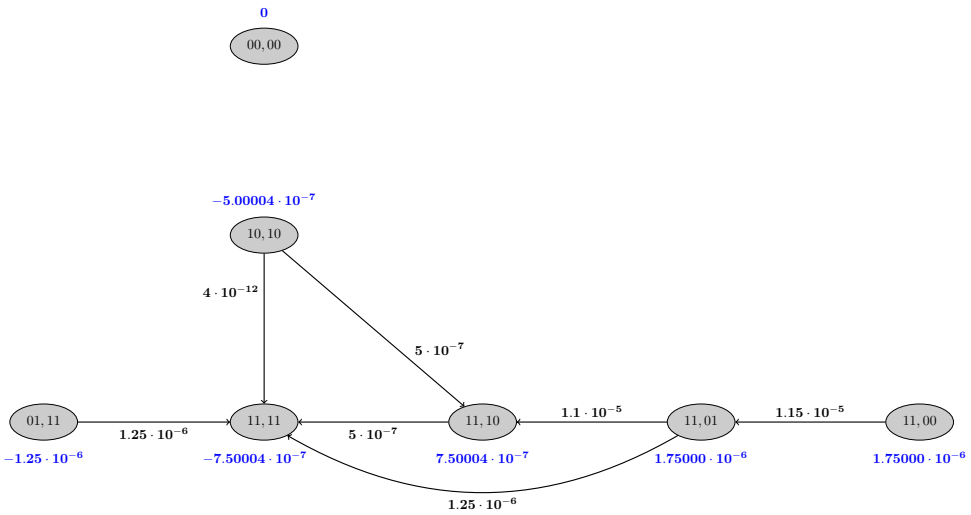

Fig. 7. Intermediate weighted error graph after selecting vertex $(00, 00)$ as codeword

maximum rank $rank(00, 00) = 1.15003 \cdot 10^{-5}$ and is selected as codeword. After removing the edges connected to or from one of the complementary vertices $CV(00, 00) = \{(00, 01), (00, 10), (00, 11)\}$ and removing the vertices $CV(00, 00)$ itself the resulting error graph is shown in Fig. 7. The intermediate error graph from Fig. 7 shows that the vertices $(11, 01)$ and $(11, 00)$ has both the maximum rank of $1.75000 \cdot 10^{-6}$ and we can choose which one we want to select as codeword. After selecting one of them and removing all complementary vertices as well as connecting edges the error detecting code e.g. $C = \{(00, 00), (01, 11), (10, 10), (11, 00)\}$ is determined and the check bit functions are well defined. The resulting error graph has no connecting edge and can therefore detect all modeled errors.

The second case is similar with the difference that only errors are considered which change codewords in codewords of the inner code C_{in}. Errors which are detected by the outer code C_{out} can be removed from the initial error graph by removing vertices $(v_1, v_2, \ldots, v_{n'}, c_1, c_2, \ldots, c_m)$ where the first n' bits don't reflect the outer code $v \notin C_{out}$.

IV. Implementation of the Heuristic

The implementation details of the heuristic to assign one additional check bit is described in this section. To find suitable data structures for the graph is essential because the memory used by the heuristic algorithm increases exponential with the

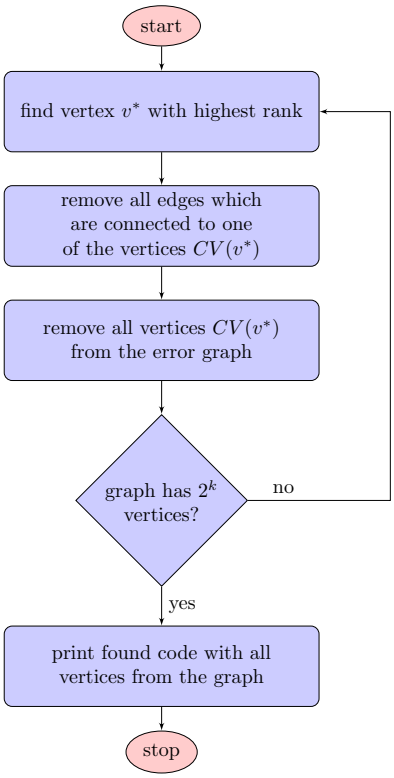

Fig. 5. Flow chart of the heuristic

block length n of the code. The vertices can be described as an unsigned integer. So the initial graph representing the fault model to define an check bit for data bits consists of all vertices which can be represented as the integer interval from 0 to 2^n.

If we can find an error pattern which describes the analyzed errors representing this error pattern in an data structure and calculating the number of edges by vertices can save a lot of memory compared to a adjacent matrix. For example if we want to find a code for all 2 bit errors for two data bits and one check bit as shown in Fig. 8 one error pattern could be the three error vectors $(0,1,1)$, $(1,0,1)$, $(1,1,0)$ in comparison the adjacent matrix for a code with block length 3 would have $2^3 \cdot 2^3 = 64$ entries. The error pattern for a unidirectional fault model could also include the error vectors, but a edge between two vertices v_1, v_2 with error vector e only exists if the condition $v_1 \wedge e = 0$ or the condition $v_2 \wedge e = 0$ is satisfied. A error graph is typically a sparse graph where the number of edges is much smaller then the maximum number 2^{2n}. So if we can not find an error pattern and have to save all edges within the memory a adjacency list seems appropriate.

To allow a effective calculation of the ranks the relevant sum of weights $weight(v)$ per vertex can be cached. The cache may be defined for the interval of the vertices and may contain markers (e.g. a negative value) showing that the vertex of this entry is already removed. So when the algorithm remove a vertex and all connected edges, only the entries of the cache which were connected to the vertex have to be updated and the entry for the vertex itself has to be marked as removed. For small block length n the cache should match the size of vertices to allow a fast calculation. If the array can't fit into the memory, the cache size can be adjusted and the state of the graph may be saved with a bit field representing which vertices are removed.

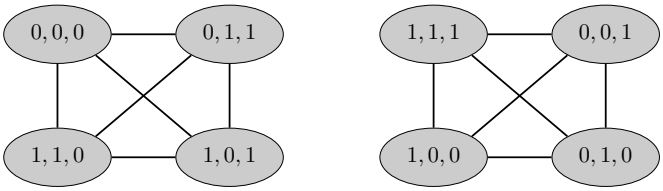

Fig. 8. Error graph for 2 bit errors and block codes of length 3

V. Error Detection for Multi-Level NAND Flash Memories

In this section the quality of the heuristic is evaluated by comparing the codes found by the heuristic with well-known error detecting codes. As an example application we used multi-level NAND flash memories with retention errors modeled according to [2]. The error graph for two flash cells are shown in Fig. 4 and described at the end of section II.

Unidirectional and asymmetric error detection codes for memories which can store $q > 2$ values are defined and operate within the residue class ring $\mathbb{Z}/q\mathbb{Z}$. In practical applications only $q = 2^a$ values for $a \in \{2, 3\}$ are used and block lengths are only defined for integer values of $K = k/a$ data symbols and $M = m/a$ check symbols. The binary encoding from Table I is used to interpret the amount of stored electrons of a flash cell. A binary codeword $v = (u_1, u_2, \ldots, u_k, c_1, c_2, \ldots, c_m)$ with k data bits and m check bits is interpreted within the quotient ring $\mathbb{Z}/2^a\mathbb{Z}$ as a codeword $V = (U_1, U_2, \ldots, U_M, C_1, C_2, \ldots, C_M)$.

The optimal systematic code to detect all unidirectional errors in $\mathbb{Z}/q\mathbb{Z}$ needs $\lceil \log_q K(q-1)+1 \rceil$ check symbols and is computed by

$$c = \sum_{i=1}^{K} (q - 1 - U_i)$$

where $U_1, U_2, \ldots, U_K \in \mathbb{Z}/q\mathbb{Z}$ are the values stored in the data cells and the check symbols $C_1, C_2, \ldots, C_M$ are the q-ary representation of c [1]. This means in our case with $q = 2^2$ for only $K = 1$ data memory cell one redundant cell and for up to $K \leq 5$ data cells two redundant cells are enough to detect all unidirectional errors. When we use 2 redundant cells for more then $K > 5$ data cells, the code defined with the check symbols

$$c = \left(\sum_{i=1}^{K} (q - 1 - U_i) \right) \mod q^2 \qquad (1)$$

can detect up to $t = \frac{q^2}{l} - q$ asymmetric errors of maximum magnitude l [5].

TABLE III
CODES WITH ONE REDUNDANT CHECK SYMBOL TO DETECT ASYMMETRIC RETENTION ERRORS IN MLC NAND FLASH

# data symbols (K)		1	2	3	4	5	6	7
# modeled errors		48	448	3,840	31,744	258,048	2,080,768	16,711,680
error probability per word		$1.9999 * 10^{-4}$	$2.9997 * 10^{-4}$	$3.9994 * 10^{-4}$	$4.9990 * 10^{-4}$	$5.9985 * 10^{-4}$	$6.9979 * 10^{-4}$	$7.9972 * 10^{-4}$
prob. of unmodeled errors		$6 * 10^{-6}$	$9 * 10^{-6}$	$1.2 * 10^{-5}$	$1.5 * 10^{-5}$	$1.8 * 10^{-5}$	$2.1 * 10^{-5}$	$2.4 * 10^{-5}$
Heuristic	detected errors	100%	97.545%	95.990%	94.988%	94.008%	93.562%	93.786%
	prob. of det.	$1.9399 * 10^{-4}$	$2.9097 * 10^{-4}$	$3.8794 * 10^{-4}$	$4.8471 * 10^{-4}$	$5.8143 * 10^{-4}$	$6.7775 * 10^{-4}$	$7.7534 * 10^{-4}$
	prob. of undet. errors	0	$2.3498 * 10^{-10}$	$3.9603 * 10^{-9}$	$1.9282 * 10^{-7}$	$4.1990 * 10^{-7}$	$1.0394 * 10^{-6}$	$3.7569 * 10^{-7}$
Unidirectional	detected errors	100%	97.991%	96.042%	94.739%	94.048%	93.792%	93.745%
	prob. of det.	$1.9399 * 10^{-4}$	$2.9097 * 10^{-4}$	$3.8794 * 10^{-4}$	$4.8490 * 10^{-4}$	$5.8185 * 10^{-4}$	$6.7879 * 10^{-4}$	$7.7572 * 10^{-4}$
	prob. of undet. errors	0	$1.8809 * 10^{-11}$	$3.7619 * 10^{-11}$	$6.2799 * 10^{-11}$	$9.4347 * 10^{-11}$	$1.3230 * 10^{-10}$	$1.7667 * 10^{-10}$
Linear	detected errors	97.917%	95.536%	94.740%	94.229%	93.995%	93.871%	93.811%
	prob. of det.	$1.9399 * 10^{-4}$	$2.9097 * 10^{-4}$	$3.8793 * 10^{-4}$	$4.8489 * 10^{-4}$	$5.8184 * 10^{-4}$	$6.7877 * 10^{-4}$	$7.7569 * 10^{-4}$
	prob. of undet. errors	$8.8 * 10^{-11}$	$2.3520 * 10^{-9}$	$5.1269 * 10^{-9}$	$8.9672 * 10^{-9}$	$1.3873 * 10^{-8}$	$1.9843 * 10^{-8}$	$2.6878 * 10^{-8}$

First we want to compare codes with one redundant flash memory cell to detect the most common retention errors. So we can define two check bits or one check symbol in $\mathbb{Z}/4\mathbb{Z}$. Following the construction methods described in [1] and [5] we can define a check symbol

$$c = \left(\sum_{i=1}^{K} (3 - U_i) \right) \mod 4 \tag{2}$$

and use it for detecting retention errors.

Another method to protect multi-level flash cells is to use the Gray encoding from Table I to represent the stored values of the flash cells and use linear block codes to detect errors. The linear error detecting code should detect errors with a small amount of bit errors. We can define two redundant check bits with

$$\begin{aligned} c_1 &= u_1 \oplus u_2 \oplus u_3 \oplus \ldots \oplus u_k \\ c_2 &= u_1 \oplus u_3 \oplus u_5 \oplus \ldots \oplus u_{k-1} \end{aligned} \tag{3}$$

for the data bits $u_1, u_2, \ldots, u_k$ where c_1 is the overall parity bit and the XOR sum of c_2 includes only one data bit of each flash cell. The code can detect all single cell errors and multiple cell errors if the error modifies a odd number of bits.

The above two methods are compared in Table III with the experimental results of the proposed heuristic from section III. The first row shows how many data cells K are protected by one redundant flash cell, correspondingly $k = 2K$ data bits are encoded in one codeword with two check bits. In the next row we can find the amount of modeled errors where each error can be represented with an edge in an error graph. Such an error can be the result of a loss of electrons in the storage layer of one or more involved flash cells. In the third row of Table III the probability of an retention error within a codeword or in other words the involved $K + 1$ memory cells is shown. Because in the paper [2] three percent of the retention errors are not further specified, in the next row we can see the probability that an unmodeled error occurs. So the first four rows of Table III describe the used error model in our case the initial error graph.

In the next three rows (5, 6, 7) the experimental results of a error detecting code determined by the proposed heuristic are shown. The first row shows how many of the modeled errors can be detected by the used code. In the next row the probability that a modeled error occurred and will be detected is listed. And in the last row of the error detection results of the proposed heuristic the probability that an modeled error modified a codeword and can't be detected by the proposed code is shown. The unidirectional error detection codes defined by equation (2) are called unidirectional in Table III and are shown in row 8, 9 and 10. For the linear codes listed in the rows 11, 12 and 13 the stored amount of electrons of a flash cell is interpreted with the Gray code and the binary error detection code from the equations (3) are used to detect errors.

In Table III it is shown that the specialized code for unidirectional errors has the lowest probability that a modeled error occurred which can't be detected. But all the listed codes show similar good error detection probabilities for retention errors, sometimes the heuristic detect more modeled errors sometimes the unidirectional error or the linear code. If we look at the bigger picture we can see that for all analyzed codes the probability that an unmodeled error occurs is of magnitude higher than the probability that a modeled error gets undetected. The error probability of the modeled errors are very different, in the error graph with 8 flash cells the most likely error e.g. between the memory words $(0,0,0,0,0,0,0,0)$ and $(0,0,0,0,1,0,0,0)$ has the probability $p = 1.342 \cdot 10^{-6}$ and the most unlikely error between the words $(2,2,2,2,2,2,2,2)$ and $(3,3,3,3,3,3,3,3)$ has the probability $p = 2.56 \cdot 10^{-22}$. So other kinds of errors e.g. program interference errors which happens when the state of a flash cell changes while a neighboring cell is being programmed due to capacitance-coupling are more likely e.g. between the memory words $(0,0,0,0,0,0,0,3)$ and $(0,0,0,0,0,0,0,2)$ with probability $p \approx 1.5 \cdot 10^{-11}$ then some modeled errors with a very low probability. The simulations were computed at a typical

workstation with a frequency of 2.6 GHz and 6 GB Memory and for codes of block length 12 (6 flash cells) the calculation time was below 1 minute, for the block length of 16 bit the simulation took 22 hours.

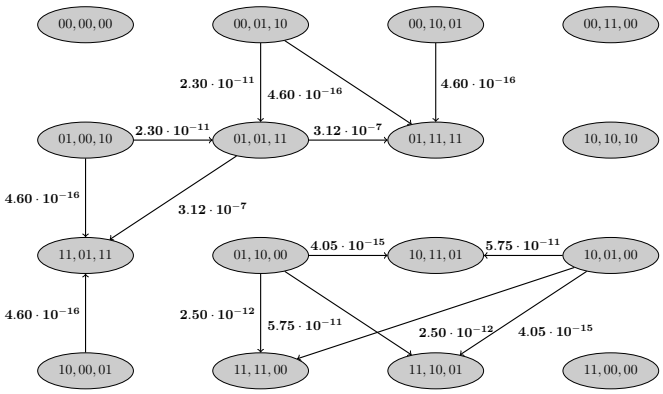

Fig. 9. Error graph of undetected errors for code determined by heuristic

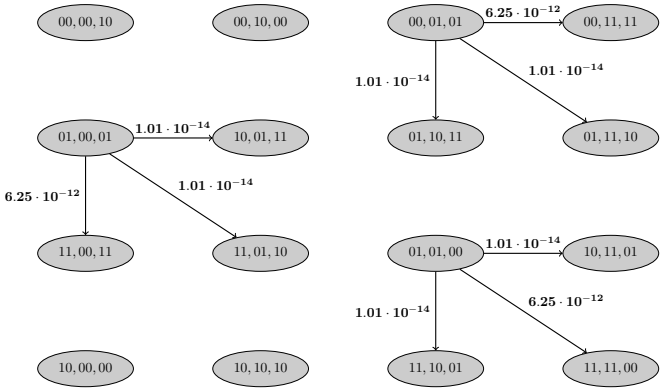

Fig. 10. Error graph of undetected errors for unidirectional error code

In Fig. 9 an error graph shows the undetected retention errors for three flash cells which are protected by an error detecting code determined by the proposed heuristic. In comparison Fig. 10 shows the error graph for undetected retention errors for three flash cells which are protected by the unidirectional error detecting code calculated by equation (2). The error graph of the unidirectional error code is more structured and only errors which affects more then one cell can not be detected.

If we want to use more then one redundant cell to protect against retention errors the unidirectional error detection code from equation (1) can detect all modeled errors.

VI. Conclusion

In this paper an error graph was introduced to model arbitrary functional errors. It was explained by examples how arbitrary combinational errors can be adequately described by the error graph. The accuracy of the unidirectional error model with a limited magnitude was compared with a error model derived form experimental results [2].

It was demonstrated how the check bits can be determined by simple graph-theoretical algorithms determining linear or nonlinear Boolean functions minimizing the number of edges in the error graph. The data structure of the algorithm was carefully chosen to allow relatively large word length of the vertices to be effectively processed.

A detailed analysis of error detecting capabilities of well-known codes and new codes derived by error graphs was shown for retention errors of multi-level NAND flash cells. It was shown that the error graph for unidirectional errors with a limited magnitude of $l = 1$ doesn't cover the error graphs of retention errors with the modeled error probabilities. But it could be shown that codes optimal for the unidirectional error model are suitable for retention errors of multi-level NAND flash cells. Codes determined by the proposed heuristic have almost the same error detection capabilities as well-known codes for special error models.

Acknowledgment

The authors would like to thank the anonymous VERFE'15 reviewers for their helpful comments on an earlier version of this work.

References

[1] B. Bose and D.K. Pradhan. Optimal unidirectional error detecting/correcting codes. *IEEE Transactions on Computers*, C-31(6):564–568, June 1982.

[2] Yu Cai, Erich F Haratsch, Onur Mutlu, and Ken Mai. Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 521–526. IEEE, 2012.

[3] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. Codes for multi-level flash memories: Correcting asymmetric limited-magnitude errors. In *IEEE International Symposium on Information Theory (ISIT) 2007*, pages 1176–1180, June 2007.

[4] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. Codes for asymmetric limited-magnitude errors with application to multilevel flash memories. *IEEE Transactions on Information Theory*, 56(4):1582–1595, April 2010.

[5] Noha Elarief, Bella Bose, and Samir Elmougy. Limited magnitude error detecting codes over z_q. *IEEE Transactions on Computers*, 62(5):984–989, 2013.

[6] T. Klove, B. Bose, and N. Elarief. Systematic, single limited magnitude error correcting codes for flash memories. *IEEE Transactions on Information Theory*, 57(7):4477–4487, July 2011.

[7] Yifat Manzor and Osnat Keren. Amalgamated q-ary codes for multi-level flash memories. In *IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) 2012*, pages 98–103. IEEE, 2012.

[8] G. Nieß, T. Kern, and M. Gössel. Determination of almost optimal check bits for an arbitrary error model. *11th International Workshop on Boolean Problems*, pages 155–163, 2014.