Codes for Partially Stuck-at Memory Cells

Antonia Wachter-Zeh and Eitan Yaakobi
Department of Computer Science
Technion—Israel Institute of Technology, Haifa, Israel
Email: {*antonia, yaakobi*}*@cs.technion.ac.il*

***Abstract*—This paper studies a new model of stuck-at memory cells which is motivated by the defect model of multi-level cells in non-volatile memories such as flash memories and phase change memories. If a cell can store the levels $0, 1, \ldots, q-1$, we say that it is *partially stuck-at* s if it can only store values which are at least s, where $1 \leq s \leq q-1$. In this paper, we consider the case $s = 1$. A lower bound on the redundancy of a code which masks u partially stuck-at cells is $u(1 - \log_q(q-1))$. An upper bound of $\min\{u, n\big(1 - \log_q(q-1)\big)\}$ on the redundancy can be achieved by either codes using only positive levels or codes which mask stuck-at cells (not partially). Our main contribution in this paper is the construction of efficient codes which improve upon this upper bound on the redundancy for all values of q and u.**

***Index Terms*—defect cells, flash memories, (partially) stuck-at cells, phase change memories**

I. Introduction

Non-volatile memories such as flash memories and phase change memories (PCMs) have paved their way to be a dominant memory solution for a wide range of applications, from consumer electronics to solid state drives. The rapid increase in the capacity of these memories along with the introduction of multi-level technologies have significantly reduced their cost. However, at the same time, their radically degraded reliability demands for advanced signal processing and coding solutions.

PCM consists of cells that can be in distinct physical states. In the simplest case, a PCM cell has two possible states, an amorphous state and a crystalline state. Multi-level PCM cells can be designed by using partially crystalline states [3]. Failures of PCM cells stem from the heating and cooling processes of the cells. These processes may prevent a PCM cell to switch between its states and thus the cells become *stuck-at* [7], [10], [17], [19]. Similarly, in the multi-level setup, the cells can get stuck-at one of the two extreme states, amorphous or crystalline. Or, alternatively, the cells cannot be programmed only to one of the two states, but can be in all other ones; for example, if a cell cannot be programmed to the amorphous state it can still be at the crystalline state as well as all other intermediate ones. A similar phenomenon might appear in flash memories. Here, the cells are programmed by electrically charging them with electrons in order to represent multiple levels. If charge is trapped in a cell, then its level can only be increased, or it may happen that due to defects, the cell can only represent some lower levels. Inspired by this defect model, the main goal of this paper is the study of codes which combat cells that are *partially stuck-at*. In [6], codes for unreliable cells in flash memories were studied, i.e., the charge level in certain cells should not exceed a threshold to prevent these cells from causing many errors. This model can be seen as the dual of partially stuck-at cells.

In the binary version of stuck-at cells, the memory consists of n binary cells and at most u of them are stuck-at either in the zero or one state. The encoder knows the locations and values of the stuck-at cells, while the decoder does not. The problem is to find the number of messages that can be encoded and successfully decoded by the decoder. If u cells are stuck-at then it is not possible to encode more than $n-u$ bits, and thus the problem is to design codes with redundancy close to u. The same problem is relevant for the non-binary setup of this model, where the cells can be stuck-at any level.

The study of codes for memories with stuck-at cells, also known as *memories with defects*, takes a while back to the 1970s. To the best of our knowledge, the problem was first studied in 1974 by Kuznetsov and Tsybakov [14]. Since then, several more papers have appeared, e.g. [1], [2], [4], [9], [12], [13], [18], [21], [22], [23]. The main goal in all these works was to find constructions of codes which mask a fixed number of stuck-at cells and correct another fixed number of additional arbitrary errors. Recently, the connection between memories with stuck-at cells and the failure models of PCM cells has attracted a renewed attention to this prior work. Several more code constructions as well as efficient encoding and decoding algorithms for the earlier constructions were studied; see e.g. [5], [11], [15], [16], [20].

The main focus of this paper is the study of codes which mask cells that are *partially stuck-at*. We assume that cells can have one of the q levels $0, 1, \ldots, q-1$. Then, it is said that a cell is *partially stuck-at* level s, where $1 \leq s \leq q-1$, if it can only store values which are at least s. We assume in this work that cells can only be partially stuck-at at level $s = 1$. We first notice that for a code masking u partially stuck-at level 1 cells, its redundancy has to be at least $u(1 - \log_q(q-1))$. On the other hand, codes which mask any number of partially stuck-at cells can simply be constructed by using only the positive levels in each cell. This trivial construction has redundancy $n(1-\log_q(q-1))$. Alternatively, codes which mask u stuck-at cells (not partially) can be used to mask u partially stuck-at cells. However, the redundancy of such a code will be at least u symbols. Thus, these two schemes give an upper bound on the redundancy of codes for masking partially stuck-at cells, and if one assumes the existence of optimal codes which mask u stuck-at cells then it is possible to achieve redundancy $\min\{u, n\big(1 - \log_q(q-1)\big)\}$. In this work, we design codes which mask partially stuck-at cells

ISBN 978-3-8007-3659-1

with smaller redundancy and analyze how far they are from the lower bound. In particular, for $u < q$, we show that one redundancy symbol is sufficient to mask all partially stuck-at cells. If $u = q$, we give two constructions. The first one uses matrices with special properties to construct such codes, and the second one uses *binary* codes which mask stuck-at cells (not partially). Then, it is shown how the latter one can be extended for all other values where $u > q$.

The rest of the paper is organized as follows. In Section II, we introduce notations and formally define the model of partially stuck-at cells studied in the paper. In Section III, we propose codes along with encoding and decoding algorithms for the case $u < q$. In Section IV, we give a construction for the setup $u = q$ and in Section V we show a more general solution for $u \geq q$ by using codes which mask binary stuck-at cells. Finally, Section VI concludes this paper.

II. Definitions and Preliminaries

In this section, we formally define the models of stuck-at cells studied in the paper and introduce the notations and tools we will use in the sequel. The cells are assumed to represent q-ary symbols having values belonging to the set $\{0, 1, \ldots, q-1\}$. In general, for a positive integer n, we denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$.

There are two main models for stuck-at memory cells. In the first one, it is said that a cell is ***stuck-at level*** $s \in [q]$ if it can store only the value s. In the second one, a cell is said to be ***partially stuck-at level*** $s \in [q]$ if it can store only values which are at least s. In this work, when studying partially stuck-at cells, we only consider $s = 1$ and thus whenever saying that a ***cell is partially stuck-at*** or mention a ***partially stuck-at cell***, we assume that it is partially stuck-at level 1.

We use the notation $(n, M)_q$ to indicate a code of length n over the alphabet $[q]$ with M codewords. Its redundancy is denoted by $r = n - \log_q M$. If the code is linear over $[q]$ (in which case q is a power of a prime and $[q]$ corresponds to the finite field $\mathbb{F}_q$ of order q), then we denote it by $[n, k]_q$, where k is its dimension and its redundancy is $r = n - k$. Whenever we speak about a linear $[n, k, d]_q$ code, then d refers to the minimum Hamming distance of an $[n, k]_q$ code. We also denote by $\rho(n, d, q)$ the smallest redundancy of any linear code of length n and minimum Hamming distance d over $\mathbb{F}_q$.

Definition 1 *We define the following code properties:*

1) *An $(n, M)_q$ code $\mathcal{C}$, **which can mask u stuck-at cells**, is a coding scheme with encoder $\mathcal{E}$ and decoder $\mathcal{D}$. The input to the encoder $\mathcal{E}$ is the locations and values of some $u' \leq u$ stuck-at cells and a message $m \in [M]$. Its output is a vector $\mathbf{y}_m \in [q]^n$ which matches the values of the u' stuck-at cells and its decoded value is m, that is $\mathcal{D}(\mathbf{y}_m) = m$.*
2) *An $(n, M)_q$ code $\mathcal{C}$, **which can mask u partially stuck-at cells**, is a coding scheme with encoder $\mathcal{E}$ and decoder $\mathcal{D}$. The input to the encoder $\mathcal{E}$ is the locations of some $u' \leq u$ partially stuck-at cells and a message $m \in [M]$. Its output is a vector $\mathbf{y}_m \in [q]^n$ which its value on these locations is at least 1 and its decoded value is m, that is $\mathcal{D}(\mathbf{y}_m) = m$.*

Similar definitions follow if the code $\mathcal{C}$ is linear.

In the design of codes which mask (partially) stuck-at cells, the goal is to optimize the redundancy of such codes, while providing efficient encoding and decoding algorithms. For positive integers n, q, u, where $u \leq n$, we denote by $r(n, q, u)$ the minimum redundancy of codes over $[q]$ which mask u partially stuck-at cells. A trivial lower bound on $r(n, q, u)$ is given in the next lemma.

Lemma 1 (Lower Bound) *For any $(n, M)_q$ code which masks u partially stuck-at cells, the redundancy is at least*

$$r(n, q, u) \geq u(1 - \log_q(q-1)).$$

Proof: The $n-u$ cells which are not partially stuck-at can each carry one q-ary information symbol and the u partially stuck-at cells can still represent $q-1$ possible values (all values except for 0). Hence, we can store at most $M \leq q^{n-u}(q-1)^u$ q-ary vectors and the code redundancy satisfies

$$r \geq n - \log_q(q^{n-u}(q-1)^u) = u(1 - \log_q(q-1)),$$

which implies that $r(n, q, u) \geq u(1 - \log_q(q-1))$. ∎

A trivial construction to mask any $u \leq n$ partially stuck-at cells is to use only the values $1, 2, \ldots, q-1$ as possible symbols in any cell. Any cell therefore stores $\log_q(q-1)$ q-ary information symbols. The achieved redundancy is

$$n - \log_q\left((q-1)^n\right) = n\big(1 - \log_q(q-1)\big),$$

and therefore $r(n, q, u) \leq n\big(1 - \log_q(q-1)\big)$.

Furthermore, notice that every code which can mask u stuck-at cells can also mask u partially stuck-at cells. This provides another upper bound on the value of $r(n, q, u)$. Codes for this model were studied before and one such construction is stated in the following theorem; see e.g. [9].

Theorem 1 (Codes for Stuck-at Cells) *Let $\mathcal{C}$ be an $(n, M)_q$ code with minimum distance $u+1$, where q is a prime power. Then, the code $\mathcal{C}$ can be used to mask u stuck-at cells.*

It is not completely known whether this construction is optimal with respect to its achieved redundancy. The redundancy of such a code is at least u since there are u stuck-at cells that cannot store information. However, we assume that optimal codes masking stuck-at cells with redundancy u exist. We carry this assumption since our goal is to construct codes for partially stuck-at cells with redundancy smaller than u.

To summarize, we conclude with the following bounds on the value of $r(n, q, u)$ for all positive integers n, q, u where $u \leq n$:

$$u(1-\log_q(q-1)) \leq r(n, q, u) \leq \min\{u, n(1-\log_q(q-1))\}. \quad (1)$$

These bounds will serve as a reference for our constructions in the paper, as we should ensure to construct codes with better redundancy than the upper bound and then study how far their redundancy is from the lower bound.

ISBN 978-3-8007-3659-1

III. Construction for $u < q$ Partially Stuck Cells

A. Code Construction

In this section, we show a simple construction for masking any $u < q$ partially stuck-at cells, which works for any q (not necessarily a prime power). Let us illustrate the idea of this construction with an example.

Example 1 *Let $u = 2$, $q = 3$, $n = 5$, and $\mathbf{H} = (1\ 1\ 1\ 1\ 1)$. We will show how to construct a $(5, 3^4 = 81)_3$ code which masks any $u' \leq u$ partially stuck-at cells.*

Let $\mathbf{m} = (m_0\ m_1\ m_2\ m_3) = (2\ 0\ 1\ 0)$ be the information we want to store and assume that the two cells at positions $\phi_0 = 1$ and $\phi_1 = 2$ are partially stuck-at. Further, define $\mathbf{w} = (0\ m_0\ m_1\ m_2\ m_3) = (0\ 2\ 0\ 1\ 0)$. Given $\mathbf{w}$, ϕ_0 and ϕ_1, our goal is to find a vector $\mathbf{x}$ such that $\mathbf{y} \equiv (\mathbf{w} + \mathbf{x}) \bmod 3$ masks the partially stuck-at cells and yet the information stored in $\mathbf{m}$ can be reconstructed from $\mathbf{y}$.

We use $\mathbf{x} = z \cdot \mathbf{H}$, for some $z \in \mathbb{F}_3$. Since $w_1 = 2$, we can choose any value from $\{0, 1, 2\}$ for z except for 1, and since $w_2 = 0$, we can choose any value but 0. Thus, we choose $z = 2$ and encode the vector $\mathbf{y}$ to be $\mathbf{y} = \mathbf{w} + 2 \cdot (1\ 1\ 1\ 1\ 1) = (2\ 1\ 2\ 0\ 2)$, which masks the two partially stuck-at cells.

To reconstruct $\mathbf{w}$, given $\mathbf{y}$, we notice that $y_0 = z = 2$ and we can simply calculate $\mathbf{y} - y_0 \cdot \mathbf{H}$ to obtain $\mathbf{w}$ and therefore $\mathbf{m}$.

Thus, the required redundancy for masking $u = 2$ partially stuck-at cells is $r = 1$ (the first symbol). The lower bound from (1) is $2 \cdot (1 - \log_3 2) = 0.738$, while the upper bound is $\min\{2, 5 \cdot (1 - \log_3 2)\} = 1.845$.

The following theorem shows that with the $1 \times n$ matrix $\mathbf{H} = (1\ 1\ \ldots\ 1)$ from Example 1, we can always mask any $u' \leq u$ partially stuck-at cells as long as $u < q$.

Theorem 2 (Construction for $u < q$) *If $u < q$ and $u \leq n$, then for all n, there exists an $(n, M = q^{n-1})_q$ code which masks u partially stuck-at cells and has redundancy one.*

Proof: Let $\phi_0, \phi_1, \ldots, \phi_{u-1}$ be the positions of some u partially stuck-at cells and let $\mathbf{m} = (m_0, m_1, \ldots, m_{n-2}) \in [q]^{n-1}$ be the message to be written to the memory $\mathbf{w}$ of length n. We first set $\mathbf{w} = (0\ m_0\ m_1\ \ldots\ m_{n-2})$. Let $\mathbf{H} = (1\ 1\ \ldots\ 1)$ be a $1 \times n$ matrix and we perform all calculations modulo q. Since $u < q$, there exists at least one value $v \in [q]$ such that $w_{\phi_0}, w_{\phi_1}, \ldots, w_{\phi_{u-1}} \neq v$. Choose $z = q - v \equiv -v$ and therefore, $(w_{\phi_i} + z) \neq 0 \equiv q$, $\forall i \in [u]$. Therefore, $\mathbf{w} + (q - v) \cdot \mathbf{H}$ masks all u partially stuck-at cells.

Such a matrix $\mathbf{H}$ exists for any length n, and since we need only one symbol to store $z = -v$, the redundancy is one. ■

This principle works for *any* n since $\mathbf{H}$ always exists. Further, q does not have to be a prime power and we can use this strategy even if $[q]$ does not constitute a finite field.

Theorem 2 provides a significant improvement compared to codes for the (stronger) model of usual stuck-at cells, where the required redundancy is at least u. In order to compare this to the second term in the upper bound, we will use the approximation $\ln(q) - \ln(q-1) \approx 1/q$ for q large enough and thus,

$$(1 - \log_q(q-1)) \approx \frac{1}{q \ln q}.$$

Hence, for $n \geq q \ln q$, the construction from Theorem 2 achieves better (i.e., smaller) redundancy than the upper bound from (1). Similarly, for q large enough, the lower bound on the redundancy is approximately $u/(q \ln q)$ and since $u < q$ this value approaches zero. This suggests that codes with better redundancy might exist. A similar analysis can be made for smaller values of q, however we omit the details due to the lack of space.

B. Encoding and Decoding Algorithm

Algorithms 1 and 2 show the encoding and decoding procedures for partially stuck-at cells with the construction from Theorem 2 and $u < q$.

Algorithm 1. Encoding$\big(\mathbf{m}; \{\phi_0, \ldots, \phi_{u-1}\}\big)$

Input: • message: $\mathbf{m} = (m_0\ m_1\ \ldots m_{n-2}) \in [q]^{n-1}$
• positions of stuck-at cells: $\{\phi_0, \ldots, \phi_{u-1}\} \subseteq [n]$

1 $\mathbf{w} = (w_0\ w_1\ \ldots\ w_{n-1}) \leftarrow (0\ m_0\ m_1\ \ldots m_{n-2})$
2 Set v such that $v \notin \{w_{\phi_0}, w_{\phi_1}, \ldots, w_{\phi_{u-1}}\}$
3 $z \leftarrow q - v$
4 $\mathbf{y} \leftarrow \mathbf{w} + z \cdot (1\ 1\ \ldots\ 1)$

Output: vector $\mathbf{y}$ with $y_{\phi_i} \in \{1, \ldots, q-1\}$, $\forall i \in [u]$

The encoder proceeds as in the proof of Theorem 2 and the decoder simply uses the fact that the first position is used to store the value of z.

Algorithm 2. Decoding$\big(\mathbf{y}\big)$

Input: • stored vector: $\mathbf{y} = (y_0\ y_1\ \ldots y_{n-1}) \in [q]^n$

1 $\widehat{z} \leftarrow y_0$
2 $\widehat{\mathbf{m}} \leftarrow (y_1\ \ldots\ y_{n-1}) - \widehat{z} \cdot (1\ 1\ \ldots\ 1)$

Output: message vector $\widehat{\mathbf{m}} \in [q]^{n-1}$

IV. Constructions for $u = q$ Partially Stuck Cells

The next example shows that we cannot use the strategy from Theorem 2 (and Algorithms 1, 2) if $u \geq q$ in order to mask the partially stuck-at cells.

Example 2 *Let $u = q = 3$, $n = 5$, and let $\mathbf{H} = (1\ 1\ 1\ 1\ 1)$. Let $\mathbf{m} = (1\ 2\ 2\ 0)$ be the message vector we want to store and assume that the cells at positions $0, 1, 3$ are partially stuck. As in Algorithm 1, let $\mathbf{w} = (0\ 1\ 2\ 2\ 0)$.*

For $z = 0$, we obtain $\mathbf{y} = \mathbf{w}$, which does not mask the partially stuck-at cell at position 0; for $z = 1$, we obtain $\mathbf{y} = \mathbf{w} + (1\ 1\ 1\ 1\ 1) = (1\ 2\ 0\ 0\ 1)$, which does not mask the cell at position 3 and for $z = 2$, we have $\mathbf{y} = \mathbf{w} + 2 \cdot (1\ 1\ 1\ 1\ 1) = (2\ 0\ 1\ 1\ 2)$, where the stuck-at cell at position 1 is not masked.

Thus, for any $z \in \mathbb{F}_3$, there is always one value in $\mathbf{y}$ that does not match the partially stuck-at cells (i.e., it has value 0).

Additionally, the lower bound from Lemma 1 shows that $r(5,3,3) \geq 1.107$ *and therefore we need more than one redundancy symbol. The upper bound gives* $r \leq \min\{3, 1.845\}$.

The following theorem shows a construction based on the parity-check matrix of linear codes for $u = q$ partially stuck-at memory cells. The proof uses Theorem 5 from the appendix.

Theorem 3 (Construction for $u = q$) *Let q be a prime power, $u = q$, and let $\mathbf{H}$ be the $(n-k) \times n$ parity-check matrix of an $[n,k,d]_q$ code, where $d \geq u$.*

Then, we can mask any $u' \leq u$ partially stuck-at cells with redundancy $r = n - k \geq u - 1$ over $\mathbb{F}_q$.

Proof: Since $\mathbf{H}$ is the parity-check matrix of an $[n,k,d]_q$ code, any $(n-k) \times u$ submatrix of $\mathbf{H}$ has rank at least $d-1 \geq u-1$ and contains no all-zero column.

The reduced row echelon (RRE) of any $(n-k) \times u$ matrix of rank $u-1$ consists of $u-1$ rows as follows (up to column permutations):

$$\begin{pmatrix} 1 & \circ & \dots & \dots & \dots & \circ \\ & 1 & \circ & \dots & \dots & \circ \\ & & 1 & \circ & \dots & \circ \\ & & & \ddots & \ddots & \circ \\ \multicolumn{2}{c}{\multirow{2}{*}{0}} & & & 1 & \circ \end{pmatrix}, \tag{2}$$

where $\circ$ is any element from $\mathbb{F}_q$. The matrix $\mathbf{H}$ contains no all-zero columns, and thus its RRE contains no all-zero columns. This holds since $\mathrm{RRE}(\mathbf{H}) = \mathbf{T} \cdot \mathbf{H}$ for some full-rank matrix $\mathbf{T}$ and therefore, for some column vector $\mathbf{b}^T$ (which denotes an arbitrary column of $\mathbf{H}$), the column vector $\mathbf{a}^T = \mathbf{T} \cdot \mathbf{b}^T$ (the corresponding column of $\mathrm{RRE}(\mathbf{H})$) is all-zero if and only if $\mathbf{b} = \mathbf{0}$.

Hence, the rightmost column in (2) has at least one non-zero element and its lowermost non-zero symbol determines which of the first $u-1$ columns is considered to be in one "block" (compare Theorem 5) with the last column. Hence, there is one "block" of length two and $u-2$ blocks of length one and (2) is a special case of (3). According to Theorem 5 from the appendix, it follows that we can mask any u partially stuck-at cells with this matrix.

The redundancy is therefore $r = n - k \geq u - 1$. ∎

With Theorem 3 we obtain the following corollary.

Corollary 1 *If $u = q$ and q is a prime power, then for all $n \geq q$, there exists a code which masks q partially stuck-at cells with redundancy $r = \rho(n,q,q)$ and therefore $r(n,q,q) \leq \rho(n,q,q)$.*

Theorem 3 shows that in many cases, we can decrease the redundancy by several q-ary symbols compared to codes for usual stuck-at cells. More specifically, according to Theorem 1, in order to construct codes which correct $u = q$ stuck-at cells, one needs to use a linear code with minimum distance $u+1$, while according to Theorem 3, it is enough to use a linear code with minimum distance u. The next example demonstrates that in many cases, this improvement can be greater than one redundancy symbol.

Example 3 *Let $u = q = 5$ and $n = 127$. In order to mask u partially stuck-at cells, we use the parity-check matrix of a $[127, 118, 5]_5$ code, which is the largest code over $\mathbb{F}_5$ for $n = 127$ and $d = 5$ and thus its redundancy is $r = 9$ (see [8]).*

In order to compare this result to $u = 5$ usual stuck-at cells, we need a code with minimum distance $d = 6$. The best known one according to [8] has parameters $[127, 115, 6]_5$ so its redundancy is $r = 12$. The second term in the upper bound has value $127 \cdot (1 - \log_5 4) = 17.6$. Therefore, we get a better code construction. However, note that it is still far from the lower bound, which has value $5 \cdot (1 - \log_5 4) = 0.69$.

Another application of Theorem 3 uses the parity-check matrix of the ternary Hamming code with parameters $[n = \frac{3^r-1}{2}, n-r, 3]_3$ in order to mask $u = 3$ partially stuck-at cells. The redundancy of this construction is $r = \log_3(2n+1)$. For the case $r = 2$, we obtain a code of length four with two redundancy symbols, while the trivial construction in the upper bound achieves a better redundancy of 1.47. The following example shows how to use the same Hamming code, but we double the code length and thus reduce the redundancy.

Example 4 *Let $u = q = 3$, and $n = 8$. We will show how to encode six ternary symbols using the matrix*

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 \end{pmatrix}.$$

Any two columns of $\left(\begin{smallmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{smallmatrix}\right)$ are linearly independent (it is the parity-check matrix of the $[4,2,3]_3$ Hamming code), and therefore, any submatrix of $\mathbf{H}$, which consists of three columns, has rank two and its RRE is of the form stated in (3)*.*

Let $\mathbf{m} = (m_0\ m_1\ m_2\ m_3\ m_4\ m_5) = (1\ 0\ 2\ 0\ 1\ 2)$ be the message vector and assume that the cells at positions $0, 2, 4$ are partially stuck. We first set the memory state be $\mathbf{w} = (0\ 0\ m_0\ m_1\ m_2\ m_3\ m_4\ m_5) = (0\ 0\ 1\ 0\ 2\ 0\ 1\ 2)$. Similar to the construction in Theorem 2, we seek to find a length-2 ternary vector $\mathbf{z} = (z_0\ z_1)$ such that the vector $\mathbf{y} = \mathbf{w} + \mathbf{z} \cdot \mathbf{H}$ masks the three partially stuck-at cells. Hence, we can choose $\mathbf{z} = (1\ 1)$ and thus, $\mathbf{y} = (1\ 1\ 2\ 1\ 1\ 2\ 1\ 2)$ masks the three partially stuck-at cells. It is possible to show that this property holds for any three partially stuck-at cells and any message vector of length six.

This provides a code of length $n = 8$ and redundancy $r = 2$ for masking any $u = 3$ partially stuck-at cells. The upper bound using the trivial construction is inferior as its value is 2.95, and this construction has a smaller redundancy even if one had an optimal code for stuck-at cells with 3 redundancy symbols. The lower bound on the redundancy is 1.107.

Note that Example 4 is not a special case of Theorem 3. The principle from Example 4 works for $u = 3$ and arbitrary q with the corresponding q-ary Hamming code. Unfortunately, it is not clear how to generalize it to arbitrary values of u or how to find other non-trivial matrices which fulfill the requirements of Theorem 5 (see appendix).

ISBN 978-3-8007-3659-1

V. Construction Using Binary Codes

In this section, we describe a construction for masking u partially stuck-at cells by means of a *binary* code. This construction works for any u; however, for $u \leq q$, the constructions from the previous sections are better. To illustrate this and our construction, we start with an example for $u = q$.

Example 5 *Let $u = q = 3$, and $n = 8$ as in Example 4. We show how to construct an $(8, 27)_3$ code that can mask three partially stuck-at cells. Let the cells at positions $\{0, 5, 6\}$ be partially stuck, the message is $\mathbf{m} = (1\ 2\ 2)$, and let $\mathbf{w} = (0\ 0\ 0\ 0\ 0\ 1\ 2\ 2)$. We want to add a codeword of a **binary** code to mask the partially stuck-at cells. We do not have to care about $w_5 = 1$ since adding 0 or 1 is unequal to 0. Thus, we can use a binary code which masks $\widetilde{u} = 2$ usual stuck-at cells. We use the parity-check matrix of an $[8, 4, 4]_2$ code, which is the largest binary code of length $n = 8$ and minimum distance $d \geq \widetilde{u} + 1 = 3$ (see [8]), and therefore we obtain*

$$\mathbf{y} = \mathbf{w} + (0\ 0\ 1\ 0) \cdot \begin{pmatrix} 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&0&0&1&1&0&0&1 \\ 0&0&0&0&1&1&1&1 \end{pmatrix} = (1\ 0\ 0\ 1\ 1\ 1\ 2\ 0),$$

which masks the three partially stuck-at cells. We can easily reconstruct $\mathbf{z} = (0\ 0\ 1\ 0)$ from $(y_0\ y_1\ y_2\ y_3) = (1\ 0\ 0\ 1)$.

Assume now, the stuck-at positions are $\{0, 5, 6\}$, $\mathbf{m} = (0\ 0\ 2\ 2)$, and $\mathbf{w} = (0\ 0\ 0\ 0\ 0\ 2\ 2)$. To minimize the number of positions which we need to mask with the binary code, we use $\widetilde{z} = 1$ to define $\widetilde{\mathbf{w}} = \mathbf{w} + \widetilde{z} \cdot (1\ 1\ \ldots\ 1) = (1\ 1\ 1\ 1\ 1\ 1\ 0\ 0)$ and we proceed as before by finding a vector $\mathbf{y}$ which masks the partially stuck-at cells. However, we need one additional redundancy symbol to store $\widetilde{z}$ (e.g. we can add $\widetilde{z}$ to y_4).

Thus, for these parameters, we need redundancy $r = 5$.

Compared to Example 4, we need higher redundancy when using the binary code, but this construction works also for any $u \geq q$ and improves in this case significantly compared to codes for usual stuck-at cells (see Example 6).

Denote by $\mathbf{w}^{(u)}$ the subvector of $\mathbf{w}$ of length u consisting only of the intended information values at the partially stuck-at positions. The principle of our construction has two steps:

1) Choose $z \in \mathbb{F}_q$ such that the number of zeros and $(q-1)$s is minimized in $\widetilde{\mathbf{w}}^{(u)} = \mathbf{w}^{(u)} + z \cdot (1\ 1\ \ldots\ 1)$. Denote the number of zeros and $(q-1)$s in $\widetilde{\mathbf{w}}^{(u)}$ by $\widetilde{u}$.
2) Use a binary code to mask these $\widetilde{u}$ usual stuck-at cells.

This idea provides the following theorem.

Theorem 4 *Let n, u, q be given (where q is not necessarily a prime power) and let $\widetilde{u} = \lfloor \frac{2u}{q} \rfloor$. Assume there exists an $[n, k, d]_2$ binary code that can mask any $\widetilde{u}$ binary stuck-at cells. Then, there exists a length-n code which can mask any $u' \leq u$ partially stuck-at cells with redundancy*

$$r = (n-k) \log_q \left(\frac{q}{\lfloor q/2 \rfloor} \right) + 1.$$

Proof: Let $\mathbf{m} \in [q]^{k-1}$ and let $\mathbf{w} = (0\ \ldots\ 0\ \mathbf{m}) \in [q]^n$.

First, we show that there exists some $z \in [q]$ such that $\widetilde{\mathbf{w}}^{(u)}$ contains at most $\widetilde{u}$ zeros and $(q-1)$s after adding $z \cdot (1\ 1\ \ldots\ 1)$. Since we add the value z to all the cells, the value $\widetilde{u}$ is equivalent to the minimum sum of any two consecutive values in $\mathbf{w}^{(u)}$. Denote $v_i = |\{j : w_j = i, j \in \{\phi_0, \ldots, \phi_{u-1}\}|$, for all $i \in [q]$. Hence, we want to minimize $v_i + v_{i+1 \bmod q}$, $i \in [q]$. Since $\sum_{i=0}^{q-1} (v_i + v_{i+1 \bmod q}) = 2u$, according to the pigeonhole principle, there exists an integer i such that $v_i + v_{i+1 \bmod q} \leq \lfloor \frac{2u}{q} \rfloor$ and therefore, we can find z such that $\widetilde{\mathbf{w}}^{(u)}$ contains at most $\widetilde{u} = \lfloor \frac{2u}{q} \rfloor$ zeros and $(q-1)$s.

Second, we treat the $\widetilde{u}$ cells of $\widetilde{\mathbf{w}}^{(u)}$ which have value 0 or $q-1$ like usual binary stuck-at cells since adding 0 or 1 to the other $u - \widetilde{u}$ cells still masks those partially stuck-at cells. Therefore, according to Theorem 1, we can use an $[n, k, d]_2$ code with $d \geq \widetilde{u} + 1$ to mask these cells, which requires redundancy $n-k$. However, since in those $n-k$ cells we only need to encode binary information, it is possible to encode $\lfloor q/2 \rfloor$ symbols per cell, so the redundancy in these cells becomes $(n-k) \log_q \left(\frac{q}{\lfloor q/2 \rfloor} \right)$. Finally, we need one additional redundancy symbol to store the value of z. ∎

The factor $\log_q \left(\frac{q}{\lfloor q/2 \rfloor} \right)$ in Theorem 4 originates from the fact that the $n-k$ redundancy symbols which stem from the binary code are *binary*, but each cell is q-ary. Thus, it is possible to store additional information in these cells. The next corollary summarizes this upper bound on the redundancy.

Corollary 2 *For all $q \leq u \leq n$ we have that*

$$r(n, q, u) \leq \rho \left(n, \left\lfloor \frac{2u}{q} \right\rfloor + 1, 2 \right) \cdot \log_q \left(\frac{q}{\lfloor q/2 \rfloor} \right) + 1.$$

Therefore, when we use a binary $[n = 2^{r_H} - 1, n - r_H, 3]_2$ Hamming code, for $q = 3$, we can mask up to $u = 4$ partially stuck-at cells (since $d = 3 \geq \lfloor \frac{2u}{q} \rfloor + 1 = 3$) with redundancy $r = \log_2(n+1) + 1$. Recall that with the principle of Theorem 3, for $q = 3$, with a ternary Hamming code, we can mask up to $u = 3$ partially stuck-at cells with redundancy $r = \log_3(2n+1)$.

Example 6 *Let $q = 3$, $u = 4$, and $n = 17$. With Theorem 4, we have $\widetilde{u} = \lfloor \frac{2u}{q} \rfloor = 2$ and we need a **binary** code of length 17 and distance $\widetilde{d} \geq 3$. The largest such code is a $[17, 13, 3]_2$ code. Therefore, the construction from Theorem 4 requires redundancy $r = 4 + 1 = 5$.*

*To mask $u = 4$ usual stuck-at cells, we need a **ternary** code of length $n = 17$ and distance $d \geq u + 1 = 5$. The largest such code is a $[17, 10, 5]_3$ code with 7 redundancy symbols.*

The trivial construction from Section II needs redundancy 6.27 and thus, we have decreased the redundancy compared to the trivial construction and to usual stuck-at cells.

Note that for $u \to n$ and for large q, the trivial construction from Section II is quite good. The construction of this section outperforms the trivial construction if $u > q$ and $n \gg u$.

VI. Conclusion

In this paper, we have presented three constructions of codes which mask partially stuck-at cells: the first one requires one redundancy symbol and masks any $u < q$ partially stuck-at cells; the second one is based on linear codes and works for $u = q$ (where q is a prime power); and the last one is based on binary linear codes and can be used for any u and q. The following table provides an overview of these constructions

ISBN 978-3-8007-3659-1

and their required redundancies. Recall that $\rho(n,d,q)$ denotes the smallest redundancy of a linear code of length n and minimum Hamming distance d over $\mathbb{F}_q$.

u	Upper bound on $r(n,q,u)$
$u<q$	1 (Theorem 2)
$u=q$	$\rho(n,q,q)$ (Theorem 3)
any u,q	$\rho\Big(n,\lfloor\frac{2u}{q}\rfloor+1,2\Big)\cdot\log_q\Big(\frac{q}{\lfloor q/2\rfloor}\Big)+1$ (Theorem 4)

Thus, we have provided codes for masking partially stuck-at cells for any set of parameters and have shown that the required redundancy significantly decreases compared to known approaches for stuck-at cells and to a trivial construction.

Acknowledgement

A. Wachter-Zeh's work was supported by a Minerva Postdoctoral Fellowship.

Appendix

Theorem 5 *Let q be a prime power and let a $\kappa\times n$ matrix $\mathbf{H}=\big(H_{i,j}\big)_{j=0,\ldots,n-1}^{i=0,\ldots,\kappa-1}$ over $\mathbb{F}_q$ be given. Then, this matrix can match any $u\le\kappa\cdot(q-1)$ partially stuck-at cells if the RRE of any $\kappa\times u$ submatrix (denoted by $\mathbf{H}^{(u)}$) has the following form (up to column permutations):*

$$\mathrm{RRE}(\mathbf{H}^{(u)})= \tag{3}$$

$$\begin{pmatrix} \underbrace{1\ \bullet\ \ldots\ \bullet}_{\le q-1} & \circ\ \circ\ \ldots\ \circ & \ldots & \ldots & \ldots\ \circ \\ & \underbrace{1\ \bullet\ \ldots\ \bullet}_{\le q-1} & \circ\ \circ\ \ldots\ \circ & \ldots & \ldots\ \circ \\ & & \underbrace{1\ \bullet\ \ldots\ \bullet}_{\le q-1} & \circ\ \ldots & \ldots\ \circ \\ & 0 & & \ddots & \vdots \\ & & & & \underbrace{1\ \bullet\ \ldots\ \bullet}_{\le q-1} \end{pmatrix},$$

where $\bullet$ has to be a non-zero element from $\mathbb{F}_q$ and $\circ$ is any element from $\mathbb{F}_q$.

Proof: Assume w.l.o.g. that the stuck-at positions are $[u]$. Further, if q is a prime *power*, we fix any mapping from the elements of the extension field $\mathbb{F}_q$ to the set of integers $[q]$.

First, assume that $\mathrm{RRE}(\mathbf{H})$ can mask any u partially stuck-at cells and let us show that then also $\mathbf{H}$ can mask any u partially stuck-at cells. By assumption, there is a vector $\mathbf{z}=(z_0\ z_1\ \ldots\ z_{\kappa-1})$ such that $\mathbf{y}=\mathbf{w}+\mathbf{z}\cdot\mathrm{RRE}(\mathbf{H})$ masks all the cells. Since $\mathbf{H}=\mathbf{T}\cdot\mathrm{RRE}(\mathbf{H})$, for some $\kappa\times\kappa$ full-rank matrix $\mathbf{T}$, the vector $\widetilde{\mathbf{z}}=\mathbf{z}\cdot\mathbf{T}^{-1}$ masks the same stuck-at cells when multiplied by $\mathbf{H}$ since $\widetilde{\mathbf{z}}\cdot\mathbf{H}=\mathbf{z}\cdot\mathbf{T}^{-1}\mathbf{H}=\mathbf{z}\cdot\mathrm{RRE}(\mathbf{H})$.

Second, we prove that $\mathrm{RRE}(\mathbf{H})$ can mask any u partially stuck-at cells. To simplify notations, assume w.l.o.g. that each "block" of $\mathrm{RRE}(\mathbf{H}^{(u)})$ has length exactly $q-1$, if is shorter it is clear that the principle also works. Similar to the proof of Theorem 2, there is (at least) one value $z_0\in\mathbb{F}_q$ such that

$$z_0\cdot H_{0,i}\neq -w_i,\ \forall i=0,\ldots,q-2, \tag{4}$$

since (4) consists of (at most) $q-1$ constraints and there are q possible values for z_0. Hence, $(z_0\ z_1\ \ldots\ z_{\kappa-1})\cdot\mathrm{RRE}(\mathbf{H}^{(u)})$ will mask the first $q-1$ partially stuck-at cells for any $z_1,\ldots,z_{\kappa-1}$ since $w_i+z_0\cdot H_{0,i}\neq 0$, for all $i=0,\ldots,q-2$.

Similarly, there is (at least) one value $z_1\in\mathbb{F}_q$ such that

$$z_1\cdot H_{1,i}\neq -(w_i+z_0\cdot H_{0,i}),\ \forall i=q-1,\ldots,2q-3,$$

and $(z_0\ z_1\ \ldots\ z_{\kappa-1})\cdot\mathrm{RRE}(\mathbf{H}^{(u)})$ masks the second $q-1$ partially stuck-at cells for any $z_2,\ldots,z_{\kappa-1}$. This principle can be continued for each block of $q-1$ cells and column permutations of $\mathrm{RRE}(\mathbf{H}^{(u)})$ do not change this strategy. ■

In contrast to Theorem 2, it is not clear if a similar statement as Theorem 5 also holds if q is not a prime power.

References

[1] I. Belov and A. Shashin, "Codes that correct triple defects in memory (in Russian)," *Prob. Inf. Transm.*, vol. 13, no. 4, pp. 62–65, 1977.

[2] J. Borden and A. Vinck, "On coding for stuck-at-defects," *IEEE Trans. Inform. Theory*, vol. 33, no. 5, pp. 729–735, Sep. 1987.

[3] G. W. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, B. Rajendran, S. Raoux, and R. S. Shenoy, "Phase change memory technology," *Journal of Vacuum Science Technology B: Microelectronics and Nanometer Structures*, vol. 28, pp. 223–262, 2010.

[4] C. L. Chen, "Linear Codes for masking memory defects," *IEEE Trans. Inform. Theory*, vol. 31, no. 1, pp. 105–106, Jan. 1985.

[5] J. Fridrich, M. Goljan, and D. Soukal, "Steganography via codes for memory with defective cells," in *Allerton Conference on Communication, Control and Computing*, Sep. 2005.

[6] R. Gabrys, F. Sala, and L. Dolecek, "Coding for unreliable flash memory cells," *IEEE Comm. Letters*, vol. 18, no. 9, pp. 1491–1494, Jul. 2014.

[7] B. Gleixner, F. Pellizzer, and R. Bez, "Reliability characterization of phase change memory," in *Europ. Phase Change Ovonics Symp.*, 2009.

[8] M. Grassl, "Bounds on the minimum distance of linear codes and quantum codes," Online available at http://www.codetables.de, 2007.

[9] C. Heegard, "Partitioned linear block codes for computer memory with stuck-at defects," *IEEE Trans. Inform. Theory*, vol. 29, no. 6, pp. 831–842, Nov. 1983.

[10] K. Kim and S. Ahn, "Reliability investigations for manufacturable high density PRAM," in *IEEE Int. Rel. Physics Symp.*, 2005, pp. 157–162.

[11] Y. Kim and B. Kumar, "Coding for memory with stuck-at defects," in *IEEE Int. Conf. Communications*, Jun. 2013, pp. 4347–4352.

[12] A. V. Kuznetsov, T. Kasami, and S. Yamamura, "An error correcting scheme for defective memory," *IEEE Trans. Inform. Theory*, vol. 24, no. 6, pp. 712–718, Nov. 1978.

[13] A. Kuznetsov, "Coding in a channel with generalized defects and random errors (in Russian)," *Probl. Inf. Transm.*, vol. 21, no. 1, pp. 28–34, 1985.

[14] A. Kuznetsov and B. Tsybakov, "Coding for memories with defective cells (in Russian)," *Probl. Inf. Transm.*, vol. 10, no. 2, pp. 52–60, 1974.

[15] L. A. Lastras-Montaño, A. Jagmohan, and M. M. Franceschini, "Algorithms for memories with stuck cells," in *IEEE Int. Symb. Inf. Theory*, Jun. 2010, pp. 968–972.

[16] L. Lastras-Montaño, A. Jagmohan, and M. Franceschini, "An area and latency assessment for coding for memories with stuck cells," in *Workshop Appl. Comm. Theo. to Em. Memory Techn.*, Dec. 2010.

[17] S. Lee, J. Jeong, T. Lee, W. Kim, and B. Cheong, "A study on the failure mechanism of a phase-change memory in write/erase cycling," *IEEE Electron. Device Letters*, vol. 30, no. 5, pp. 448–450, May 2009.

[18] V. Losev, V. Konopel'ko, and Y. Daryakin, "Double-and-triple-defect-correcting codes (in Russian)," *Prob. Inf. Transm.*, vol. 14, no. 4, pp. 98–101, 1978.

[19] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. Lacaita, and R. Bez, "Reliability study of phase-change nonvolatile memories," *IEEE Trans. Dev. Mat. Rel.*, vol. 4, no. 3, pp. 422–427, 2004.

[20] N. Seong, D. Woo, V. Srinivasan, J. Rivers, and H. Lee, "SAFER: stuck-at-fault error recovery for memories," in *MICRO-43*, Dec. 2010.

[21] B. S. Tsybakov, S. I. Gelfand, A. V. Kuznetsov, and S. I. Ortyukov, "Reliable computation and reliable storage of information," in *IEEE-USSR Workshop*, Dec. 1975.

[22] B. Tsybakov, "Defects and error correction (in Russian)," *Prob. Inf. Transm.*, vol. 11, no. 1, pp. 21–30, 1975.

[23] ——, "Group additive defect-correcting codes (in Russian)," *Prob. Inf. Transm.*, vol. 11, no. 1, pp. 111–113, 1975.