Managing Device Lifecycle: Reconfigurable Constrained Codes for M/T/Q/P-LC Flash Memories

Ahmed Hareedy, Member, IEEE, Beyza Dabak, and Robert Calderbank, Fellow, IEEE

Abstract—Flash memory devices are winning the competition for storage density against magnetic recording devices. This outcome results from advances in physics that allow storage of more than one bit per cell, coupled with advances in signal processing that reduce the effect of physical instabilities. Constrained codes are used in storage to avoid problematic patterns, and thus prevent errors from happening. Recently, we introduced binary symmetric lexicographically-ordered constrained codes (LOCO codes) for data storage and data transmission. LOCO codes are capacity-achieving, simple, and can be easily reconfigured. This paper introduces simple constrained codes that support non-binary physical gates in multi, triple, quad, and the currently-in-development penta-level cell (M/T/Q/P-LC) Flash memories. The new codes can be easily modified if problematic patterns change with time. These codes are designed to mitigate inter-cell interference, which is a critical source of error in Flash devices. The occurrence of errors is a consequence of parasitic capacitances in and across floating-gate transistors, resulting in charge propagation from cells being programmed to the highest charge level to neighboring cells being programmed to lower levels or unprogrammed/erased. This asymmetric nature of error-prone patterns distinguishes Flash memories. The new codes are called \( q \)-ary asymmetric LOCO codes (QA-LOCO codes), and the construction subsumes codes previously designed for single-level cell (SLC) Flash devices (A-LOCO codes). QA-LOCO codes work for a Flash device with any number, \( q \), of levels per cell. For \( q \geq 4 \), we show that QA-LOCO codes can achieve rates greater than \( 0.95 \log_2 q \) input bits per coded symbol. The complexity of encoding and decoding is modest, and reconfiguring a code is as easy as reprogramming an adder. Capacity-achieving rates, affordable encoding-decoding complexity, and ease of reconfigurability support the growing improvement of M/T/Q/P-LC Flash memory devices, as well as lifecycle management as the characteristics of these devices change with time, which increases their lifetime.

Index Terms—Asymmetric constrained codes, \( q \)-ary codes, lexicographic ordering, data storage, Flash memories, non-binary gates, reconfigurable codes, device lifetime.

I. INTRODUCTION

DATA storage densities are increasing rapidly as modern applications, e.g., internet of things (IoT) applications, access, process, and store more and more data. In 2015, the storage density of Flash memory devices surpassed that of magnetic recording (MR) devices. This milestone resulted from multiple advances in physics, architecture, and signal processing [1], [2]. The major advance in Flash physics was enabling more than two storage levels, i.e., more than two charge levels, per cell, and thus allowing the storage of more than one bit per cell. The major advance in Flash architecture was devising the three-dimensional vertical NAND Flash structure.

The data storage industry achieves high reliability by combining constrained codes, designed to avoid problematic patterns, with error-correction (EC) codes, designed to correct the errors that remain. Constrained systems were first discussed in [3]. Run-length-limited (RLL) codes are a class of constrained codes introduced in 1970 [4], which were first used to improve the storage density of early MR devices employing peak detection [5], [6]. Modern storage devices employ sequence estimation [7] rather than peak detection, but constrained codes are still used to improve performance [6], [8]. RLL codes also find application in optical recording [9]. When first introduced in [4], lexicographic indexing was used to encode and decode RLL codes, but this was replaced by methods based on finite-state machines (FSMs) in later work [10]. Other methods that are based on lexicographic indexing were introduced in [11] and [12], which are guided by [13]. RLL codes are typically associated with transition-based signaling [4], [5].

In level-based signaling, each symbol (or bit) is associated with a distinct level for storage or transmission. For example, in the binary case, a 0 is represented by \( A_0 \) and a 1 is represented by \( A_1 \), where \( A_0 < A_1 \), in what is called bipolar non-return-to-zero (NRZ) signaling. A binary symmetric \( S_x \)-constrained code is a code that forbids the patterns in the set \( S_x \triangleq \{010, 101, 01^20, 01^31, \ldots, 01^n0, 10^n1\} \) from appearing in any codeword, where the notation \( y^r \) refers to a sequence of \( r \) consecutive \( y \)'s. On the other hand, a binary asymmetric \( A_x \)-constrained code is a code that forbids the patterns in the set \( A_x \triangleq \{101, 10^21, \ldots, 10^n1\} \) from appearing in any codeword. Both \( S_x \)-constrained codes and \( A_x \)-constrained codes are associated with level-based signaling, which is natural for Flash memory systems.

A Flash cell is a metal-oxide-semiconductor field-effect transistor (MOSFET) with a floating gate underneath its control gate [14]. Data is stored in the form of charges (electrons)
that tunnel into the floating gate during the programming phase. The amount of charge injected into the floating gate controls the threshold of the transistor, enabling the storage of digital data. The storage capacity of the cell, in terms of bits, depends on the number of allowable charge levels (including the erasure level). In Flash devices, inter-cell interference (ICI) is one of the main sources of errors [14]. Parasitic capacitances in and across floating-gate transistors result in charge propagation from cells being programmed to the highest charge level to neighboring cells being programmed to lower levels or unprogrammed/erased.\(^1\) Thus, unintentional increases in charge values occur, resulting in errors during reading. We focus on typical Flash systems, where block erasure is adopted.

The authors of [17] and [18] introduced constrained codes to prevent the level pattern \((q-1)(q-1)\) from being written in a Flash device with \(q \geq 2\) levels per cell.\(^2\) The special case of triple-level cell (TLC) Flash devices with \(q = 8\), i.e., preventing the pattern 707, was also studied in [19]. Via extensive experiments, the authors of [20] demonstrated that for multi-level cell (MLC) Flash devices (4 levels per cell), the set of level patterns to be forbidden (contribute the most to ICI) should be \(\{303, 313, 323\}\). This set was recently generalized in [21] to \(\{(q-1)(q-1), (q-1)(q-1), \ldots, (q-1)(q-2)(q-1)\}\) for a Flash device with \(q\) levels per cell. In a simplified view, a Flash memory block is a grid of word lines and bit lines, and a cell exists at each intersection of a word line with a bit line. Even though we focus more on ICI among cells on the same bit line because of its dominance as demonstrated in [14] and [20], our proposed codes can be used to mitigate ICI among cells on the same word line as well.

In our previous work [22], we introduced capacity-achieving \(S_x\)-constrained codes, named lexicographically-ordered \(S_x\)-constrained codes (LOCO codes), that make significant MR density gains possible. LOCO codes are simple, and they can be easily reconfigured to support additional constraints. The \(A_x\)-constraint forbids ICI-causing patterns in single-level cell (SLC) Flash devices (2 levels per cell). The advantage of designing codes for asymmetric errors, rather than symmetric errors, is that it becomes possible to achieve notably higher rates. In [22], we designed capacity-achieving \(A_x\)-constrained codes, named asymmetric LOCO codes (A-LOCO codes), that offer a better rate-complexity trade-off than previous codes, and that can be easily reconfigured. We anticipate using a combination of machine learning and analysis of errors collected before the EC decoder to identify new patterns that need to be forbidden as the device ages. We see (A-)LOCO codes as a method for extending device lifetime.

In this paper, we generalize our asymmetric constrained codes in [2] to Flash devices with any number, \(q\), of levels per cell. In particular, we introduce fixed-length \(q\)-ary asymmetric LOCO codes (QA-LOCO codes) for all Flash devices. QA-LOCO codes are capacity-achieving, and we devise the encoding-decoding rule for them to offer simplicity. While available literature only focuses on the effect of ICI on adjacent cells, we handle more general constraints for higher reliability in this work. We provide insights regarding how the encoding-decoding rule can be derived directly from forbidden patterns. QA-LOCO codes are also reconfigurable because of their encoding-decoding rule. We show that QA-LOCO codes contribute to significant lifetime gains for the Flash device with rates greater than 0.95 \(\log_2 q\) input information bits per coded symbol, \(q \geq 4\), at affordable complexities. High performance EC codes are also key to such gains since they are necessary to correct errors that do not result from ICI. Furthermore, we discuss ideas to preserve high access speed. We suggest that QA-LOCO codes (with EC codes) can significantly improve the performance, i.e., increase the lifetime, of multi- \((q = 4)\) and triple- \((q = 8)\)-level cell Flash memories, and can remarkably accelerate the evolution of quad- \((q = 16)\) and penta- \((q = 32)\)-level cell Flash memories, which are the next generation in the industry.

The rest of the paper is organized as follows. In Section II, we define QA-LOCO codes and introduce their cardinality. In Section III, we derive the QA-LOCO encoding-decoding rule. In Section IV, we discuss rates and make comparisons. In Section V, we present the encoding and decoding algorithms. In Section VI, we discuss complexity, storage overhead, and reconfigurability. Section VII concludes the paper.

II. DEFINITION AND CARDINALITY

Denote a Galois field (GF) of size \(q\) by GF\((q)\). Let \(\alpha\) be a primitive element of GF\((q)\). Consequently,

\[
GF(q) \triangleq \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}.
\]

Our analysis works for any GF of size \(q\). However, we focus on \(q = 2^v\), \(v \geq 1\), because of the nature of Flash devices. We write one symbol per cell, which is equivalent to \(v = \log_2 q\) bits per cell. We define the notation \(y_d^{\alpha} \triangleq y_{r-1}y_{r-2}\cdots y_0\) to refer to a sequence of \(r\) consecutive elements that are not necessarily the same but all belong to one specific set. We define \(\delta\) and \(\delta_i,\) for any \(i,\) as elements in GF\((q)\) \(\backslash \{\alpha^{q-2}\}\).

We also define \(\delta_d^{\alpha} \triangleq \delta_1\delta_2\cdots\delta_0\) as a sequence in \([GF(q) \backslash \{\alpha^{q-2}\}]^\tau\), with \(\delta_d^{\alpha} = \delta_0 = \delta\). We now formally define QA-LOCO codes, which are \(Q_x\)-constrained.

**Definition 1:** A QA-LOCO code \(QC_{m,x}^g\) with \(q = 2, m \geq 1,\) and \(x \geq 1\) is defined by the following properties:

1. Each codeword \(c\) in \(QC_{m,x}^g\) has its symbols in GF\((q)\) and is of length \(m\) symbols.
2. Codewords in \(QC_{m,x}^g\) are ordered lexicographically.
3. Each codeword \(c\) in \(QC_{m,x}^g\) does not contain any of the patterns in the set \(Q_x^g\), where:

\[
Q_x^g \triangleq \{\alpha^{q-2}\delta_d^{\alpha^{q-2}}, \forall \delta_d^{\alpha} \notin [GF(q) \backslash \{\alpha^{q-2}\}]^\tau \mid 1 \leq \tau \leq x\}.
\]

4. The code \(QC_{m,x}^g\) contains all codewords satisfying the above three properties.

Lexicographic ordering of codewords means codewords are ordered in an ascending manner following the rule \(0 < \alpha < \cdots < \alpha^{q-2}\) for any symbol, and the symbol significance

---

\(^1\)Asymmetric errors resulting from charge leakage over time along with other problems, e.g., programming/erasing failures, in Flash devices are handled by EC techniques [15], [16].

\(^2\)Note that charge levels are directly translated to threshold voltage levels. For simplicity, levels are defined by their indices \(\{0, 1, \ldots, q-1\}\).
reduces from left to right. In particular, starting from the left, we say $c_{u_1} < c_{u_2}$ if and only if for the first symbol position the two codewords differ at, $c_{u_1}$ has a “less” symbol than that of $c_{u_2}$.

Let $c$ be an element in GF$(q)$. Define $a \triangleq L(c)$ as the Flash charge level-equivalent to symbol $c$, which is given by:

$$a \triangleq L(c) \triangleq \begin{cases} 0, & c = 0, \\ \text{gflgo}_a(c) + 1, & \text{otherwise}, \end{cases}$$

where gflgo$_a(c)$ returns the power of the GF element $c$ with gflgo$_a(1) = 0$. Thus, the set of charge levels equivalent to $c \in \{0, 1, 2, 3, \ldots, q - 1\}$, with level 0 being the erasure level, and the set of charge-level patterns equivalent to $Q^a_x$ in (1):

$$(q - 1)\mu^a_2(q - 1), \forall \mu^a_2 \in \{0, 1, 2, \ldots, q - 2\}^x,$$

where $\mu^a_2 \triangleq \{L(\delta_{r-1})L(\delta_{r-2})\ldots L(\delta_0)\}$ is defined as a sequence in $\{0, 1, 2, \ldots, q - 2\}^x$. Observe that the total number of elements in $Q^a_x$ is:

$$|Q^a_x| = \frac{(q - 1) + (q - 1)^2 + \cdots + (q - 1)^x}{q - 2}.$$  

Observe also that in the case of $x = 1$, the set in (1) reduces to $Q^a_1 = \{\alpha^{q - 2}0\alpha^{q - 2}, \alpha^{q - 2}1\alpha^{q - 2}, \ldots, \alpha^{q - 2}(q - 3)\alpha^{q - 2}\}$, with $|Q^a_1| = q - 1$ as confirmed by (4). The set of level patterns equivalent to $Q^a_1$ is $\{(q - 1)(q - 1)(q - 1)(q - 1)(q - 1)(q - 1)\}$, which is the exact same set in [21], and also in [20] for $q = 4$. It is clear that for the binary case $(q = 2)$, $Q^a_2$ is simply $A_x$ (in this case, the result of (4) is $x = l$ the Hopital’s rule).

Table I lists all the codewords and cardinalities of the QA-LOCOS codes $\mathbb{C}^2_{m,1}$ with $m \in \{1, 2, 3, 5\}$. These codes have $q = 2$ (binary) and $x = 1$, i.e., they are the A-LOCOS codes $\mathbb{C}_{l,1}$ with $m \in \{1, 2, 3, 5\}$ [2]. For QA-LOCOS codes with $q > 2$, it is not feasible to present similar tables because the number of codewords is too large.

The partition of QA-LOCOS codewords into groups is essential to deriving the cardinality and later the decoding-decoding rule. We partition the codewords in $\mathbb{C}^q_{m,x}$, $m \geq 2$, into three groups according to what they start with from the left, i.e., at the left-most symbols (LMSs), as follows:

**Group 1:** Codewords starting with $\delta, \delta \in \text{GF}(q) \setminus \{\alpha^{q-2}\}$, from the left, i.e., at the LMS.

**Group 2:** Codewords starting with $\alpha^{q-2}\alpha^{q-2}$ from the left, i.e., at the LMSs.

**Group 3:** Codewords starting with $\alpha^{q-2}\delta^{x+1}$, $\delta^{x+1} \in [\text{GF}(q) \setminus \{\alpha^{q-2}\}]^x$, from the left, i.e., at the LMSs.

In the case of $2 \leq m \leq x + 1$, there exist only $(q - 1)^{m-1}$ codewords in Group 3, which have fewer symbols than the aforementioned $x + 2$ LMSs. The following analysis also applies for such codewords.

Observe that given the set of forbidden patterns $Q_x^a$ in (1), there are no other symbol options for a codeword in $\mathbb{C}^q_{m,x}$ to have at its LMSs. Now, we are ready to enumerate QA-LOCOS codewords recursively.

**Theorem 1:** The cardinality (size) of a QA-LOCOS code $\mathbb{C}^q_{m,x}$, denoted by $N_q(m, x)$, is given by:

$$N_q(m, x) = qN_q(m - 1, x) - (q - 1)N_q(m - 2, x) + (q - 1)^xN_q(m - x - 2, x), \quad m \geq 2,$$

where the defined cardinalities are:

$$N_q(m, x) \triangleq (q - 1)^m, \quad m \leq 0, \quad \text{and} \quad N_q(1, x) \triangleq q.$$  

**Proof:** We use the group structure stated above to prove the recursive formula (5).

**Group 1:** Each codeword in Group 1 in $\mathbb{C}^q_{m,x}$ starts with $\delta, \delta \in \text{GF}(q) \setminus \{\alpha^{q-2}\}$, from the left, and therefore corresponds to a codeword in $\mathbb{C}^q_{m-1,x}$ such that they share the $m - 1$ right-most symbols (RMSs). This correspondence is surjective. Since $\delta$ is in $\{0, 1, \alpha, \ldots, \alpha^{q-3}\}$, the correspondence is $q - 1$ codewords of length $m$ to 1 codeword of length $m - 1$. Thus, the cardinality of Group 1 in $\mathbb{C}^q_{m,x}$ is given by:

$$N_{q,1}(m, x) = (q - 1)N_q(m - 1, x).$$

**Group 2:** Each codeword in Group 2 in $\mathbb{C}^q_{m,x}$ starts with $\alpha^{q-2}\alpha^{q-2}$ from the left, and therefore corresponds to a codeword in $\mathbb{C}^q_{m-1,x}$ that starts with $\alpha^{q-2}$ from the left such that they share the $m - 2$ RMSs. This correspondence is bijective. The codewords in $\mathbb{C}^q_{m-1,x}$ that start with $\alpha^{q-2}$ from the left are obtained by excluding the codewords in $\mathbb{C}^q_{m-1,x}$ that start with $\delta, \delta \in \text{GF}(q) \setminus \{\alpha^{q-2}\}$, from the left (the codewords of Group 1 in $\mathbb{C}^q_{m-1,x}$) from all the codewords in $\mathbb{C}^q_{m-1,x}$. Thus, the cardinality of Group 2 in $\mathbb{C}^q_{m,x}$ is given by:

$$N_{q,2}(m, x) = N_q(m - 1, x) - N_{q,1}(m - 1, x) = N_q(m - 1, x) - (q - 1)N_q(m - 2, x),$$

where the second equality in (8) is reached aided by (7) to compute $N_{q,1}(m - 1, x)$.

**Group 3:** Each codeword in Group 3 in $\mathbb{C}^q_{m,x}$ starts with $\alpha^{q-2}\delta^{x+1}$, $\delta^{x+1} \in [\text{GF}(q) \setminus \{\alpha^{q-2}\}]^{x+1}$, from the left, and therefore corresponds to a codeword in $\mathbb{C}^q_{m-1,x}$ that starts with $\delta, \delta \in \text{GF}(q) \setminus \{\alpha^{q-2}\}$, from the left such that they share the $m - x - 2$ RMSs. This correspondence is surjective. Since $\delta$, for any $i$, is in $\{0, 1, \ldots, \alpha^{q-3}\}$, the correspondence is $\prod_{i=0}^{m-x-1}(q - 1) = (q - 1)^x$ codewords (each $\delta$ requires $\times (q - 1)$) of length $m$ to 1 codeword of length $m - x - 1$. The codewords in $\mathbb{C}^q_{m-1,x}$ that start with $\delta$ from the left are the codewords of Group 1 in $\mathbb{C}^q_{m-1,x}$. Thus, the cardinality of Group 3 in $\mathbb{C}^q_{m,x}$ is given by:

$$N_{q,3}(m, x) = (q - 1)^xN_{q,1}(m - x - 1, x) = (q - 1)^xN_q(m - x - 2, x),$$

where the second equality in (9) is reached aided by (7) to compute $N_{q,1}(m - x - 1, x)$.

---

3QA-LOCOS codes with $q = 2$ and $x = 1$ find application in magnetic recording systems adopting the extended partial-response target EEPRe4 [2], [8] in addition to SLC Flash memory systems.
TABLE I
THE CODEWORDS OF FIVE QA-LOCO CODES, QC\(^{q}_{m,1}\), \(q = 2, m \in \{1, 2, \ldots, 5\}\), AND \(x = 1\). THE THREE DIFFERENT GROUPS OF CODEWORDS ARE SHOWN FOR THE CODE QC\(^{2}_{3,1}\)

| Codeword index \(g(c)\) | Codewords of the code QC\(^{2}_{3,1}\) |
|-------------------------|----------------------------------|
|                         | \(m = 1\)                        | \(m = 2\)                        | \(m = 3\)                        | \(m = 4\)                        | \(m = 5\)                        |
| 0                       | 0                                | 00                               | 000                              | 0000                             | 00000                            |
| 1                       | 1                                | 01                               | 001                              | 0001                             | 00001                            |
| 2                       | 10                               | 010                              | 011                              | 0011                             | 00011                            |
| 3                       | 11                               | 110                              | 0110                             | 00110                            | 000110                           |
| 4                       | 4                                | 100                              | 0110                             | 00110                            | 000110                           |
| 5                       | 5                                | 110                              | 01110                            | 10000                            | 100001                           |
| 6                       | 6                                | 1110                             | 01110                            | 100001                           |                                  |
| 7                       | 7                                | 11110                            | 11000                            |                                  |                                  |
| 8                       | 8                                | 11111                            | 11100                            |                                  |                                  |
| 9                       | 9                                | 11111                            | 11100                            |                                  |                                  |
| 10                      | 10                               | 11111                            | 11100                            |                                  |                                  |
| 11                      | 11                               | 11111                            | 11100                            |                                  |                                  |
| 12                      | 12                               | 11111                            | 11100                            |                                  |                                  |
| 13                      | 13                               | 11111                            | 11100                            |                                  |                                  |
| 14                      | 14                               | 11111                            | 11100                            |                                  |                                  |
| 15                      | 15                               | 11111                            | 11100                            |                                  |                                  |
| 16                      | 16                               | 11111                            | 11100                            |                                  |                                  |
| 17                      | 17                               | 11111                            | 11100                            |                                  |                                  |
| 18                      | 18                               | 11111                            | 11100                            |                                  |                                  |
| 19                      | 19                               | 11111                            | 11100                            |                                  |                                  |
| 20                      | 20                               | 11111                            | 11100                            |                                  |                                  |

Now, the cardinality of QC\(^{3}_{m,x}\) is computed as follows using (7), (8), and (9):

\[
N_q(m, x) = \sum_{l=1}^{3} N_{q,l}(m, x)
= qN_q(m-1, x) - (q-1)N_q(m-2, x)
+ (q-1)x^{x-1}N_q(m-x-2, x),
\]

which completes the proof.

**Remark 1:** The corner case for Group 2 in QC\(^{3}_{m,x}\) is the case of \(m = 2\). We know that the cardinality of Group 2 in this case is \(N_{2,2}(2, x) = 1\) since the group only has the codeword \(a^{q-2}x^{x-2}\). From (8) in the proof of Theorem 1 and also (6):

\[
N_{2,2}(2, x) = N_{2}(1, x) - (q-1)N_{2}(0, x)
= q - (q-1) = 1,
\]

which is consistent with what we know. Furthermore, the corner case for Group 3 in QC\(^{3}_{m,x}\) is the case of \(2 \leq m \leq x+2\). We know that the cardinality of Group 3 in this case is \(N_{3,3}(m, x) = (q-1)^{m-1}\). From (9) in the proof of Theorem 1 and also (6):

\[
N_{3,3}(m, x) = (q-1)x^{x+1}N_q(m-x-2, x)
= (q-1)x^{x+1}(q-1)^{m-x-2}
= (q-1)^{m-1}, \quad -x \leq m-x-2 \leq 0,
\]

which is consistent with what we know. Note that there is no corner case for Group 1.

Observe that substituting \(q = 2\) in (5) and (6) yields:

\[
N_{2}(m, x) = 2N_{2}(m-1, x) - N_{2}(m-2, x) + N_{2}(m-x-2, x), \quad m \geq 2,
\]

where the defined cardinalities are:

\[
N_{2}(m, x) \triangleq 1, \quad m \leq 0, \quad \text{and} \quad N_{2}(1, x) \triangleq 2.
\]

These are the same cardinality equations of an A-LOCO code \(AC_{m,x}\), which is QC\(^{2}_{m,x}\), as derived in [2].

**Example 1:** Consider the QA-LOCO codes QC\(^{4}_{m,1}\) \((q = 4\) and \(x = 1)\) with \(m \in \{2, 3, \ldots, 6\}\). From (6), the defined cardinalities needed here are:

\[
N_4(-1, 1) \triangleq 3^{-1}, \quad N_4(0, 1) \triangleq 1, \quad \text{and} \quad N_4(1, 1) \triangleq 4.
\]

The cardinalities of the aforementioned QA-LOCO codes are:

\[
N_4(2, 1) = 4N_4(1, 1) - 3N_4(0, 1) + 9N_4(-1, 1) = 16,
N_4(3, 1) = 4N_4(2, 1) - 3N_4(1, 1) + 9N_4(0, 1) = 61,
N_4(4, 1) = 4N_4(3, 1) - 3N_4(2, 1) + 9N_4(1, 1) = 232,
N_4(5, 1) = 4N_4(4, 1) - 3N_4(3, 1) + 9N_4(2, 1) = 889,
N_4(6, 1) = 4N_4(5, 1) - 3N_4(4, 1) + 9N_4(3, 1) = 3409.
\]

Theorem 1 is a key result in the analysis of QA-LOCO codes. The theorem provides insights regarding the codewords of a QA-LOCO code of a specific length. As we shall see shortly, Theorem 1 and the insights it provides are fundamental to the derivation of the encoding-decoding rule, to the rate discussion, and to the algorithms.

### III. QA-LOCO Encoding-Decoding Rule

Now, we derive a formula that relates the lexicographic index of a QA-LOCO codeword to the codeword itself. We call this formula the encoding-decoding rule of QA-LOCO codes since it is the foundation of the QA-LOCO encoding and decoding algorithms presented in Section V.

We define a QA-LOCO codeword of length \(m\) symbols as \(c \triangleq c_{m-1}c_{m-2} \ldots c_0\) in QC\(^{q}_{m,x}\). The index of a QA-LOCO codeword \(c\) in QC\(^{q}_{m,x}\) is denoted by \(g(q, m, x, c)\), which is sometimes abbreviated to \(g(c)\) for simplicity. Our lexicographic index \(g(c)\) is in \(\{0, 1, \ldots, N_q(m, x) - 1\}\). For
each symbol \( c_i \), we define its level-equivalent \( a_i \triangleq L(c_i) \) as shown in (2). We also define \( a_i \triangleq 0 \) and \( a_i \triangleq 0 \), for \( i \geq m \). The same notation applies for a QA-LOCO codeword of length \( m+1 \), \( c' \) in \( QC'_{m+1,x} \), and a QA-LOCO codeword of length \( m-x \), \( c'' \) in \( QC_{m-x,x} \).

For each codeword symbol \( c_i \), define **Condition (a)** as the condition that \( c_{i+k} \cdots c_{i+2}c_{i+1} = \alpha q^{-2} \sigma_{k-1}^{q^i} \in \left[ GF(q) \setminus \{ q^i \} \right]_{k-1}^{\sigma_{k-1}^{q^i}} \), for some \( k \in \{ 1, 2, \ldots \} \). Condition (a) can also be written as \( a_{i+k} \cdots a_{i+2}a_{i+1} = (q-1)m_{k-1} \), \( m_{k-1} \in \{ 0, 1, 2, \ldots, q-2 \}^{k-1} \), for some \( k \in \{ 1, 2, \ldots \} \). For example, for a QA-LOCO code with \( q = 4 \), \( m \geq 7 \), and \( x = 3 \), if we have \( c_{6}c_{5}c_{4}c_{3} = \alpha^2q^1 \) then, \( k_5 = 1 \), \( k_4 = 2 \), and \( k_3 = 3 \).

The following theorem introduces the encoding-decoding rule of QA-LOCO codes. Observe that indexing is straightforward for the case of \( q = 2 \).

**Theorem 2:** Consider a QA-LOCO code \( QC'_{m,x} \) with \( m \geq 2 \). Let \( c \) be a QA-LOCO codeword in \( QC'_{m,x} \). The relation between the lexicographic index \( g(c) \) of this codeword and the codeword itself is given by:

\[
g(c) = \sum_{i=0}^{m-1} a_i(q-1)^{\gamma_i} N_q(i - \gamma_i, x),
\]

where \( \gamma_i \) for symbol \( c_i \) is computed as follows:

\[
\gamma_i = \begin{cases} x - k_i + 1, & k_i \text{ satisfying (a)} \text{ exists,} \\ 0, & \text{otherwise.} \end{cases}
\]

Starting from the left (LMS), parameter \( k_i \in \{ 1, 2, \ldots \} \), if exists, represents the backward distance in symbols from \( c_i \) to the nearest \( \alpha q^{-2} \) symbol. Note that \( \gamma_{m-1} = 0 \).

**Proof:** We prove Theorem 2 by induction.

**Base:** The base case is the case of \( m = 2 \). Using (5) and (6), the number of codewords in \( QC_{2,x} \) is:

\[
N_q(2, x) = qN_q(1, x) - (q-1)N_q(0, x) + (q-1)x+1N_q(-x, x) = q^2 - (q-1) + (q-1) = q^2.
\]

These \( q^2 \) codewords are in lexicographic order: 00, 01, 10, 11, 0q, 1q, \ldots, followed by \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), followed by \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \), \( \alpha q^{-2} \). We want to prove that the index obtained from (14) for each codeword \( c = c_1c_0 \) in \( QC_{2,x} \) matches its index in the aforementioned order.

First, consider the codewords in \( QC_{2,x} \) that start with \( \delta, \delta \in GF(q) \setminus \{ \alpha q^{-2} \} \), from the left, i.e., \( c = \delta \alpha q \). Since \( \gamma_1 = \gamma_0 = 0 \) from (15), (using (14) and (6)) for such codewords gives:

\[
g(c) = a_1 N_q(1, x) + a_0 N_q(0, x) = L(\delta) q + L(c_1),
\]

which is indeed the correct indexing formula. The reason is that for \( m = 2 \), no codewords are removed. Thus, an increment by 1 in \( L(c_1) \) results in adding \( g \) to the index, and an increment by 1 in \( L(c_0) \) results in adding 1 to the index. For example, consider the case of \( q = 4 \). The codeword 11 is the 5th in order. From (17), \( g(c) = L(1) \times 4 + L(1) = 4 + 1 = 5 \). The codeword \( \alpha \alpha q^2 \) is the 11th in order. From (17), \( g(c) = L(\alpha) \times 4 + L(\alpha^2) = 8 + 3 = 11 \).

Second, consider the codewords in \( QC'_{2,x} \) that start with \( \alpha q^{-2} \) from the left, i.e., \( c = \alpha q^{-2} \). For \( c_1, \gamma_1 = 0 \) from (15). For \( c_0, k_0 = 1 \), and therefore from (15), \( \gamma_0 = x \). Using (14) and (6) for such codewords gives:

\[
g(c) = a_1 N_q(1, x) + a_0 (q-1)^{\gamma} N_q(-x, x) = L(\alpha q^{-2}) q + L(c_0) = (q-1)q + L(c_0),
\]

which is indeed the correct indexing formula because of the same reason in the previous paragraph. For example, consider the case of \( q = 4 \). The codeword \( \alpha^2 q \) is the 14th in order. From (18), \( g(c) = 3 \times 4 + L(\alpha) = 12 + 2 = 14 \). Note that from (6), \( N_q(1, x) \equiv q \) and \( N_q(0, x) \equiv 1 \), for all \( x \in \{ 1, 2, \ldots \} \).

**Assumption:** We assume that (14) is true for all the QA-LOCO codes \( QC_{m,x} \), \( m \in \{ 2, 3, \ldots, m \} \). Mathematically, we assume the following:

\[
g(q, m, x, c) = \sum_{i=0}^{m-1} a_i(q-1)^{\gamma_i} N_q(i - \gamma_i, x),
\]

where \( \gamma \) is in \( QC_{m,x} \). The symbols of \( \gamma \) are \( \gamma_i, i \in \{ 0, 1, \ldots, m-1 \} \). For each \( \gamma_i, \gamma_i = L(\gamma_i) \) is its level-equivalent defined as in (2), and \( \gamma_i \) is defined as in (15).

To be proved: We want to prove that given the base and the assumption, (14) is also true for the QA-LOCO code \( QC'_{m+1,x} \). In particular, we want to prove that:

\[
g(q, m+1, x, c') = \sum_{i=0}^{m} a_i(q-1)^{\gamma_i} N_q(i - \gamma_i, x),
\]

where \( \gamma_i \) is defined for each \( c_i \) as in (15), and it is a function of \( x \) and \( k_i \) that depends on symbols to the left of \( c_i \).

We reuse our group structure to prove (20). We prove that (20) is true for the three groups in the QA-LOCO code of length \( m + 1 \), which means it is true for the entire code. Note that our group structure can be defined for a QA-LOCO code of any length. We also reuse the codeword correspondence from the proof of Theorem 1, with \( m + 1 \) replacing \( m \).

**Group 1:** The codewords in Group 1 in \( QC'_{m+1,x} \) start at index 0, and the same applies for the corresponding codewords in \( QC_{m,x} \) (recall the lexicographic ordering rule from the start of Section II). The correspondence here is surjective. Thus, for this group, the shift in codeword indices between \( c' \) in \( QC'_{m+1,x} \) and the corresponding \( c \) in \( QC_{m,x} \) depends on the value of \( \delta, \delta \in GF(q) \setminus \{ \alpha q^{-2} \} \), at the LMS \( c_m' \) of \( c' \). In particular,

\[
g(q, m+1, x, c') - g(q, m, x, c) = L(c_m') N_q(m, x).
\]

For example, if \( c_m' = 0 \), the shift has to be 0, while if \( c_m' = \alpha \), the shift has to be 2N_q(m, x). Next, using (19):

\[
g(q, m+1, x, c') = a_m' N_q(m, x) + \sum_{i=0}^{m-1} a_i(q-1)^{\gamma_i} N_q(i - \gamma_i, x).
\]

Observe that \( \gamma_i = 0 \), and because \( c_m' \neq \alpha q^{-2}, \gamma_i = 0 \) from (15). On the other hand, \( \gamma_i = 0 \). Since \( \gamma_i \) and \( c_i \)
share the $m$ RMSs and $\gamma_{m-1} = \gamma_{m-1}$, (22) can be written as:
\[
g(q, m + 1, x, c') = a'_m(q - 1)\gamma_{m-1}N_q(m - \gamma'_{m-1}, x) + \sum_{i=0}^{m-1} a'_i(q - 1)\gamma_{i}N_q(i - \gamma'_i, x).
\]
Consequently, we get:
\[
g(q, m + 1, x, c') = \sum_{i=0}^{m} a'_i(q - 1)\gamma_{i}N_q(i - \gamma'_i, x).
\]

**Group 2:** The codewords in Group 2 in $Q_{m+1,x}$ start right after Groups 1 and 3 in $Q_{m+1,x}$, and the corresponding codewords in $Q_{m,x}$ start right after Group 1 in $Q_{m,x}$ (recall the lexicographic ordering rule from the start of Section II). Moreover, the correspondence here is surjective. Thus, for this group, the shift in codeword indices between $c'$ in $Q_{m+1,x}$ and the corresponding $c''$ in $Q_{m,x}$ depends on the values in the sequence $\delta_1, \delta_2 \in [GF(q) \setminus \{\alpha\}^x]$, which follows the symbol $c'_m = \alpha q^{2} - 1$ (the LMS), at $c'_{m-1}, c'_{m-2}, \ldots, c'_{m-x}$ of $c'$. At each symbol $c'_{m-j}$, $j \in \{1, 2, \ldots, x\}$, an additional shift of $L(c'_{m-j})(q - 1)^{x-j}N_q(m - x, x)$ should be added. Putting all terms together results in:
\[
g(q, m + 1, x, c') = \sum_{i=0}^{m} a'_i(q - 1)\gamma_{i}N_q(i - \gamma'_i, x).
\]

Next, using (19) and also (7) to compute $N_q(m + 1, x)$ and $N_q(m - x, x)$, we get:
\[
g(q, m + 1, x, c') = (q - 1)N_q(m, x) + \sum_{j=1}^{x} L(c'_{m-j})(q - 1)^{x-j}N_q(m - x, x).
\]

Consequently, we get:
\[
g(q, m + 1, x, c') = \sum_{i=0}^{m} a'_i(q - 1)\gamma_{i}N_q(i - \gamma'_i, x).
\]

**Group 3:** The codewords in Group 3 in $Q_{m+1,x}$ start right after Group 1 in $Q_{m+1,x}$, and the corresponding codewords in $Q_{m-x,x}$ start at index 0 (recall the lexicographic ordering rule from the start of Section II). The correspondence here is surjective. Thus, for this group, the shift in codeword indices between $c'$ in $Q_{m+1,x}$ and the corresponding $c''$ in $Q_{m-x,x}$ depends on the values in the sequence $\delta_1, \delta_2 \in [GF(q) \setminus \{\alpha\}^x]$, which follows the symbol $c'_m = \alpha q^{2} - 1$ (the LMS), at $c'_{m-1}, c'_{m-2}, \ldots, c'_{m-x}$ of $c'$. At each symbol $c'_{m-j}$, $j \in \{1, 2, \ldots, x\}$, an additional shift of $L(c'_{m-j})(q - 1)^{x-j}N_q(m - x, x)$ should be added. Putting all terms together results in:
\[
g(q, m + 1, x, c') = \sum_{i=0}^{m} a'_i(q - 1)\gamma_{i}N_q(i - \gamma'_i, x).
\]
and \(\gamma'_{m-x-1} = \gamma'_{m-x-1}\), (32) can be written, aided by (33), as:

\[
g(q, m + 1, x, c') = a'_m (q - 1)^{\gamma'_{m-x}} N_q (m - \gamma'_m, x) \\
+ \sum_{i = m - x}^{m - 1} a'_i (q - 1)^{\gamma'_i} N_q (i - \gamma'_i, x) \\
+ \sum_{i = 0}^{m - 2} a'_i (q - 1)^{\gamma'_i} N_q (i - \gamma'_i, x).
\]

Consequently, we get:

\[
g(q, m + 1, x, c') = \sum_{i = 0}^{m - 2} a'_i (q - 1)^{\gamma'_i} N_q (i - \gamma'_i, x).
\]

(35)

From (24), (30), and (35), (20) is proved for all three groups in \(\mathcal{Q}C^q_{m+1,x}\), which means (20) is proved for the entire code. This completes the proof by induction, and thus, the encoding-decoding rule in (14) is proved for any QA-LOCO code \(\mathcal{Q}C^q_{m,x}\) with \(q \geq 2, m \geq 2,\) and \(x \geq 1\).

Remark 2: Consider a QA-LOCO code \(\mathcal{Q}C^q_{m,x}\) with \(2 \leq m \leq x + 2\). We expect to have the following index for any codeword \(\mathbf{c} = \alpha q^{-2} g_{d_0}^{m-2}, g_{d_0}^{m-1} \in [\text{GF}(q) \setminus \{\alpha^{q-2}\}]^{m-1}\), in Group 3 in this code:

\[
g(\mathbf{c}) = (q - 1) N_q (m - 1, x) + \sum_{i = 0}^{m - 2} a_i (q - 1)^{\gamma_i}.
\]

(36)

From Theorem 2, the index of a codeword \(\mathbf{c}\) in Group 3 in this code is:

\[
g(\mathbf{c}) = \sum_{i = 0}^{m - 2} a_i (q - 1)^{\gamma_i} N_q (i - \gamma_i, x) \\
= (q - 1) N_q (m - 1, x) \\
+ \sum_{i = 0}^{m - 2} a_i (q - 1)^{x + i - m - 2} N_q (m - x - 2, x).
\]

(37)

Observe that here \(\gamma_i = x - k_i + 1 = x - (m - 1 - i) + 1 = x + i - m + 2\). Since \(-x \leq m - x - 2 \leq 0\), we know from (6) that \(N_q (m - x - 2, x) = (q - 1)^{m - x - 2}\). Thus, (37) becomes (36), confirming what we expect.

Observe that substituting \(q = 2\) in (14) yields:

\[
g(\mathbf{c}) = \sum_{i = 0}^{m - 1} a_i N_q (i - \gamma_i, x).
\]

(38)

where for \(\gamma_i \neq 0\), i.e., \(\gamma_i \neq 0\), \(\gamma_i\) here is either \(x\) in the case of \(c_{i+1} = 1\) or \(0\) in the case of \(c_{i+1} = 0\). Thus, \(\gamma_i\) can be written as \(L(\gamma_i - 1) = x = a_{i+1} x\). Substituting \(\gamma_i = a_{i+1} x\) in (38) gives the rule of an A-LOCO code \(\mathcal{A}C_{m,x}\) (binary), which is \(\mathcal{Q}C^q_{m,x}\), as derived in [2].

Example 2: We use (14) to compute the index of two QA-LOCO codewords in \(\mathcal{Q}C^4_{6,2}\) (\(q = 4, m = 6,\) and \(x = 2\)). Using Theorem 1, the required cardinalities are \(N_q (1, 2) = 3, N_q (0, 2) = 1, N_q (1, 2) = 4, N_q (2, 2) = 16, N_q (3, 2) = 61, N_q (4, 2) = 223,\) and \(N_q (5, 2) = 817\).

The first example codeword is the 334th codeword \(\mathbf{c} = 011a^20x0a\), where the order was obtained by generating all codewords in \(\mathcal{Q}C^4_{6,2}\). This codeword has \(a_5 = 0, a_4 = a_3 = 1, a_2 = 3, a_1 = 0,\) and \(a_0 = 2\). From (15), we get \(\gamma_5 = \gamma_4 = \gamma_3 = \gamma_2 = 0,\) \(\gamma_1 = x = 2,\) and \(\gamma_0 = x - 1 = 1\). Thus, from (14):

\[
g(\mathbf{c}) = \sum_{i = 0}^{5} a_i (3^{\gamma_i}) N_q (i - \gamma_i, 2) \\
= N_q (4, 2) + N_q (3, 2) + 3N_q (2, 2) + 6N_q (1, 2) \\
= 223 + 61 + 3 \times 16 + 6 \times 3^{-1} = 334,
\]

which is the correct index.

The second example codeword is the 1850th codeword \(\mathbf{c} = a_0 a^2 a^2 a_0\), where the order was obtained by generating all codewords in \(\mathcal{Q}C^4_{6,2}\). This codeword has \(a_5 = 2, a_4 = 0, a_3 = a_2 = 3, a_1 = 2,\) and \(a_0 = 0\). From (15), we get \(\gamma_5 = \gamma_4 = \gamma_3 = 0,\) \(\gamma_2 = \gamma_1 = x = 2,\) and \(\gamma_0 = x - 1 = 1\). Thus, from (14):

\[
g(\mathbf{c}) = \sum_{i = 0}^{5} a_i (3^{\gamma_i}) N_q (i - \gamma_i, 2) \\
= 2N_q (5, 2) + 3N_q (3, 2) + 27N_q (0, 2) + 18N_q (1, 2) \\
= 2 \times 817 + 3 \times 61 + 27 \times 1 + 18 \times 3^{-1} = 1850,
\]

which is the correct index.

Theorem 2 is the key result behind the simple, reconfigurable QA-LOCO encoding and decoding we offer. The theorem provides one-to-one mapping from an index to the corresponding codeword, which is the encoding, and one-to-one demapping from a codeword to the corresponding index, which is the decoding. Section V provides algorithms for QA-LOCO encoding and decoding, and Section VI provides a discussion of their reconfigurability.

We now develop insights about how the encoding-decoding rule in (14) can be directly derived from the set of forbidden patterns. Consider the symbol \(c_i\) of the codeword \(\mathbf{c}\) in \(\mathcal{Q}C^q_{m,x}\). We want to compute the incremental contribution of \(c_i\) to the overall index \(g(\mathbf{c})\). Denote this contribution by \(g_i(c_i)\). Following [13], this contribution can be viewed as the number of codewords in \(\mathcal{Q}C^q_{m,x}\) starting with \(c_{m-1} c_{m-2} \cdots c_{i+1}\) from the left and preceding the first codeword starting with \(c_{m-1} c_{m-2} \cdots c_{i+1} c_i\) from the left according to the lexicographic ordering.

If \(c_i = 0\), then \(g_i(c_i) = 0\). If \(c_i \neq 0\), then we have two scenarios. The first scenario is when Condition (*) is not satisfied, resulting in \(\gamma_i = 0\). In this case, the contribution \(g_i(c_i)\) equals the number of codewords in \(\mathcal{Q}C^q_{i+1,x}\) (of length \(i + 1\)) that start with a symbol \(\psi, L(\psi) < L(c_i)\), from the left. Thus, the contribution is:

\[
g_i(c_i) = L(c_i) N_q (i, x) = N_q (i, x), \quad \gamma_i = 0.
\]

(39)

The second scenario is when Condition (*) is satisfied, resulting in \(\gamma_i \neq 0\). In this case, the contribution \(g_i(c_i)\) equals the number of codewords in \(\mathcal{Q}C^q_{i+1,x}\) (of length \(i + 1\)) that start with \(\psi \delta^\gamma_{\gamma_i}, L(\psi) < L(c_i)\) and \(\delta^\gamma_{\gamma_i} \in [\text{GF}(q) \setminus \{\alpha^{q-2}\}]^{\gamma_i}\), from the left in order that the constraint is satisfied. Thus,
the contribution is:
\[
g_i(c_i) = L(c_i) \left[ \prod_{j=i-\gamma_i}^{i-1} (q - 1) \right] N_q(i - \gamma_i, x) = a_i(q - 1)^{\gamma_i} N_q(i - \gamma_i, x), \quad \gamma_i \neq 0. \tag{40}
\]
This contribution of \( c_i \) in (39) and (40) to the index \( g(c) \) is perfectly consistent with (14).

The discussed insights are quite useful in understanding the encoding-decoding rule of QA-LOCO codes given in (14). Furthermore, these insights can lead to a general method for designing any constrained code based on lexicographic indexing once the set of patterns to forbid is known, which is part of our future work.

IV. ACHIEVABLE RATES AND COMPARISONS

Before we introduce the achievable rates of QA-LOCO codes and make comparisons with other codes, we first discuss how to perform bridging and achieve self-clocking.

Bridging is required in order to prevent forbidden patterns from appearing while transitioning from a codeword into the next one [22]. Consider the QA-LOCO code \( QC_{q,m}^c \) \((q = 4, m = 5, \text{ and } x = 1)\). Assume that we are about to write the following two consecutive codewords on an MLC Flash device (4 levels per cell): 01\( \alpha_2^2 \alpha_2^2 \) and \( 1\alpha_2^4 \) (01). The stream containing the two consecutive codewords to be written on ten consecutive cells is 01\( \alpha_2^2 \alpha_2^2 \)102001, and it does contain the forbidden pattern \( \alpha_2^2 \alpha_2^2 \). Bridging fixes such a problem.

Let \( e \triangleq \alpha_q^{-2} \). We perform bridging in a QA-LOCO code \( QC_{q,m,x}^c \) via adding bridging patterns as follows:

1) If the RMS of a codeword and the LMS of the next codeword are both \( \alpha_q^{-2} \)'s, bridge with \( e \), i.e., bridge with \( x \) consecutive \( e \) symbols (\( x \) consecutive cells programmed to level \( q - 1 \)).
2) Otherwise, bridge with \( 0^x \), i.e., bridge with \( x \) consecutive 0 symbols (\( x \) consecutive unprogrammed cells).

Applying this bridging method to the above scenario results in the following stream 01\( \alpha_2^2 \alpha_2^2 \)01\( \alpha_2^4 \) (01). Bridging with 0 between the two codewords prevents the forbidden pattern from appearing across the codewords.

Our bridging is not only simple, but also optimal in the sense that it provides the maximum protection from ICI for the symbols at the edges of QA-LOCO codewords. Note also that this bridging helps us reduce the number of codewords to be removed from the QA-LOCO code such that we achieve self-clocking to only two codewords as we discuss below.\(^4\)

Self-clocking is required in order to maintain calibration of the system [2], [6]. Self-clocked constrained codes do not allow long streams of the same symbol to be written (transmitted). Given our bridging method illustrated above for a QA-LOCO code \( QC_{q,m,x}^c \), even if we repeat a same-symbol codeword consecutive times in a stream, as long as this symbol is in \( GF(q) \backslash \{0, \alpha_q^{-2}\} \), bridging will guarantee that two transitions to then from a different symbol (0) occur right before each new codeword in the stream. This does not happen with only two same-symbol codewords, which are 0\( ^m \) and \( e^m \). Consequently, these are the only codewords we need to remove from \( QC_{q,m,x}^c \) to achieve self-clocking.

Definition 2: Let \( QC_{m,x}^q \) be a QA-LOCO code with \( q \geq 2, m \geq 1, \text{ and } x \geq 1 \). A self-coded QA-LOCO code (CA-LOCO code) \( QC_{m,x}^{q,c} \) is obtained from \( QC_{m,x}^q \) as follows:\(^5\)
\[
QC_{m,x}^{q,c} \triangleq QC_{m,x}^q \backslash \{0^m, e^m\}, \quad e \triangleq \alpha_q^{-2}.
\]
Therefore, the cardinality of the CQA-LOCO code is:
\[
N_q^c(m, x) = N_q(m, x) - 2. \tag{42}
\]

Define \( k_{\text{eff}}^c \) as the maximum number of consecutive cells between two consecutive transitions (all programmed to the same level or all unprogrammed) after a stream of CQA-LOCO codewords separated by bridging patterns is written; one symbol per cell. Thus, \( k_{\text{eff}}^c \) is the length of the longest run of consecutive 0’s, 1’s, \( \alpha_2^i \)'s, …, or \( \alpha_q^{-2} \)'s in a stream of CQA-LOCO codewords separated by bridging patterns. The following is one scenario under which \( k_{\text{eff}}^c \) is achieved:
\[
\delta_1 e^{m-1} - e^x - e^{m-1} \delta_0, \delta_0, \delta_1 \in GF(q) \backslash \{\alpha_q^{-2}\}. \text{ As a result, } k_{\text{eff}}^c \text{ is given by:}
\]
\[
k_{\text{eff}}^c = 2(m - 1) + x, \tag{43}
\]
which is the same equation satisfied by LOCO codes [22] and A-LOCO codes [2]. The parameter \( k_{\text{eff}}^c \) is the parameter characterizing the self-clocking property of a QA-LOCO code.

Table II lists all the messages the CQA-LOCO code \( QC_{5,1}^{c,2} \) \((q = 2, m = 5, \text{ and } x = 1)\) encodes and their corresponding CQA-LOCO codewords. The two codewords 0\( ^m \) and 1\( ^m \) are not listed. Observe that the CQA-LOCO code \( QC_{5,1}^{c,2} \) is the self-clocking A-LOCO code (CA-LOCO code) \( AC_{5,1}^{c,2} \) [2]. Again, we could not provide a similar table for CQA-LOCO codes with \( q > 2 \) because of the large number of codewords.

Now, we are ready to discuss the achievable rates of QA-LOCO codes. Consider a CQA-LOCO code \( QC_{m,x}^{q,c} \) with \( q > 2 \) as: \(^6\)

\(^4\)With more advanced bridging for \( q > 2 \), this number can be reduced to one codeword to be removed such that we achieve self-clocking. However, the reduction from two to one practically has no effect on the rate. Moreover, the effect of this on self-clocking properties is minor.

\(^5\)Only for the case of \( q = 2 \), the CQA-LOCO code \( QC_{m,x}^{c,2} \) has \( m \geq 2 \).

### Table II

| Message | Index \( g(c) \) | Codeword \( c \) |
|---------|----------------|-----------------|
| 0000    | 1              | 00001           |
| 0001    | 2              | 00010           |
| 0010    | 3              | 00011           |
| 0100    | 4              | 00100           |
| 0101    | 5              | 00110           |
| 0110    | 6              | 00111           |
| 0111    | 7              | 01000           |
| 1000    | 8              | 01001           |
| 1001    | 9              | 01100           |
| 1010    | 10             | 01110           |
| 1011    | 11             | 01111           |
| 1100    | 12             | 10000           |
| 1101    | 13             | 10001           |
| 1110    | 14             | 10010           |
| 1111    | 15             | 10011           |
|         | 16             | 11000           |
The reason is the higher number of omitted codewords when codewords from $\text{QC}_{\text{aq}}$ binary message in order to minimize the number of omitted in bits, of the messages. We can normalize this rate as follows:

$$R_{\text{QA-LOC0}}^n = \frac{\log_q (N_q(m, x) - 2)}{(m + x) \log_2 q},$$

where $R_{\text{QA-LOC0}}^n$ is measured in input bits per coded symbol. We can normalize this rate as follows:

$$R_{\text{QA-LOC0}} = \frac{\log_q (N_q(m, x) - 2)}{m + x},$$

This number becomes $85.9888$ when messages are binary here is $191516 - 2^{17} = 60444$. This number becomes $191516 - 4^8 = 125980$ when messages are 4-ary.

Except only the two codewords $0^m$ and $e^m$, all the code words satisfying the $Q^c$ constraint are in the CQA-LOC0 code $Q_{m,x}^{c,1}$. Additionally, the number of symbols we add for bridging is constant, which is $x$. Thus, CQA-LOC0 codes are capacity-achieving codes, i.e., the asymptotic rate of a CQA-LOC0 code matches the capacity.

Tables III and IV present the rates and the normalized rates of CQA-LOC0 codes $Q_{m,x}^{c,1}$ with $q \in \{4, 8, 16, 32\}$, various values of $m$, and $x \in \{1, 2\}$. The capacities are given in the last row of each table. The tables are generated as follows. For specific values of $q$ and $x$, we determine a set of rates based on the capacity. Then for each rate, we search until we find the smallest length $m$ resulting in reaching or exceeding such rate. This is the reason why the values of $m$ are not the same for different values of $q$ and $x$.

We compute the capacity of a $Q^c_{m,x}$-constrained code from a finite-state transition diagram (FSTD) representing the infinity of a sequence satisfying this $Q^c_{m,x}$ constraint [3]; the capacity, in input bits per coded symbol, is the base-2 logarithm of the largest real positive eigenvalue of the adjacency matrix corresponding to this FSTD. Example 4 illustrates this procedure in more detail.

**Example 3:** Consider the CQA-LOC0 code $Q_{m,x}^{c,1}$ ($q = 4$, $m = 9$, and $x = 1$). From the recursion in Theorem 1, we can reach that $N_4(0, 1) = 191518$. From (45), we get a rate of:

$$R_{\text{QA-LOC0}}^n = \frac{\log_2 (191518 - 2)}{(m + x) \log_2 q} = 1.7$$

input bits per coded symbol. From (46), the normalized rate is $1.7 / \log_2 4 = 0.85$.

Now, suppose that we want to encode non-binary messages, with their symbols defined over GF(4) here. The rate in this case becomes:

$$R_{\text{QA-LOC0}} = \frac{\log_2 (191518 - 2)}{9 + 1} = 0.8.$$
The characteristic polynomial of \( Q^1_q \) for QLC devices (resp., PLC devices). Essentially, this is redundancy is enough at length \( F \) respectively. For \( x = 2 \) (resp., 70)

\[
1 \leq h, p \leq q - 1, \text{ is the number of times state } h \text{ is connected to state } p \text{ (from } h \text{ to } p). \text{ Consequently, }
\]

\[
\mathbf{F} = \begin{bmatrix}
q - 1 & 1 & 0 \\
0 & 1 & q - 1 \\
q - 1 & 0 & 0 
\end{bmatrix}.
\]

The characteristic polynomial of \( \mathbf{F} \) is then:

\[
|\mathbf{F} - \lambda \mathbf{I}| = \lambda^3 - q \lambda^2 + (q - 1) \lambda - (q - 1)^2.
\]

Therefore, for \( q = 4 \) (resp., \( q = 8, q = 16 \), and \( q = 32 \)), the largest real positive eigenvalue of \( \mathbf{F} \) is \( \lambda_{\text{max}} = 3.8302 \) (resp., 7.8991, 15.9443, and 31.9706), and the capacity of a \( Q^1_q \)-constrained code is \( \log_2 \lambda_{\text{max}} \), which is 1.9374 (resp., 2.9817, 3.9950, and 4.9987) input bits per coded symbol. Capacities of \( Q^1_q \)-constrained and \( Q^3_q \)-constrained codes are listed in the last row of Table III and in the last row of Table IV, respectively.

Table III demonstrates that for all values of \( q \), the rates of CQA-LOCO codes with \( x = 1 \) and moderate lengths reach within only 1% from capacity; see the rates in the row right above the capacity row. Furthermore, Table IV demonstrates that for all values of \( q \), the rates of CQA-LOCO codes with \( x = 2 \) and moderate lengths reach within only 2% from capacity; see the rates in the row right above the capacity row. Most important, the tables show that CQA-LOCO codes for all values of \( q \) and \( x \) achieve normalized rates \( > 0.95 \), i.e., rates \( > 0.95 \log_2 q \) input bits per coded symbol, with only one exception, which is the case of \( q = 4 \) and \( x = 2 \). In other words, significant ICI mitigation in the Flash device can be achieved with only less than 5% redundancy, even later in the lifetime of the device when \( x \) can be raised to 2.

The two tables also show the effect of increasing \( q \) on the achievable rates. As \( q \) increases, the sufficient rate to protect the Flash device increases. Consider quad-level cell (QLC) and penta-level cell (PLC) Flash devices (\( q = 16 \) and \( q = 32 \), respectively). For \( x = 1 \), Table III shows that only about 1.9% (resp., 1.7%) redundancy is enough at length 66 symbols (resp., 70 symbols) for QLC devices (resp., PLC devices). For \( x = 2 \), Table IV shows that only about 3% (resp., 2.8%) redundancy is enough at length 73 symbols (resp., 77 symbols) for QLC devices (resp., PLC devices). Essentially, this is telling that the ICI mitigation via CQA-LOCO codes is coming almost for free with respect to redundancy. Having said that, increasing \( q \) results in an increase in the storage and complexity as we shall see in Section VI.

Next, we present brief comparisons between QA-LOCO codes and other codes designed for similar goals:

1) It is already not easy to design FSM-based binary constrained codes with rates close to capacity [5], [22]. This task becomes even more complicated in the non-binary domain as the capacity becomes higher. For example, to design an FSM-based constrained code with rate 0.9 according to [10], the FSTD under consideration needs to be raised to the power of 10, and each state in the encoding-decoding FSM will have \( 2^9 = 512 \) edges emanating from it [5], [10], resulting in massive storage and complexity associated with the encoder and the decoder. Our QA-LOCO codes offer simple encoding and decoding because of their rule, even with \( q > 2 \).

2) The authors of [4] introduced \( q \)-ary lexicographically-ordered RLL (Q-LO-RLL) codes. However, their constraints impose a minimum number of zeros between each two consecutive non-zero symbols. If applied for Flash, these constraints result in a significant rate loss, which is avoidable. In the binary case, LOCO codes were shown in [22] to offer a better rate-complexity trade-off compared with LO-RLL codes designed for the same purpose.

3) The authors of [21] introduced enumerative \( q \)-ary \( Q^2_q \)-constrained codes for Flash. While their codes are capacity-achieving and efficient, QA-LOCO codes offer simpler encoding and decoding compared with their unrank-rank approach. Additionally, the codes in [21] are only for the case of \( x = 1 \), which means QA-LOCO codes address more general constraints since they work for any \( x \geq 1 \).

4) The authors of [20] introduced a constrained coding technique for MLC Flash devices (\( q = 4 \)). They adopt the mapping-demapping: \( 0 \leftrightarrow 11, 1 \leftrightarrow 10, 2 \leftrightarrow 00, 3 \leftrightarrow 01 \). To forbid the level patterns in \{303, 313, 323\}, they write the left bits (lower page data) uncoded, and write the right bits (upper page data) coded via an RLL code with \( d = 2 \) and rate 0.5 to prevent the binary patterns in \{111, 101\}. While their coding scheme is simple, its overall normalized rate is \((1 + 0.5)/2 = 0.75\). Table III demonstrates that QA-LOCO codes with \( q = 4 \) and \( x = 1 \) forbid the level patterns in \{303, 313, 323\} for MLC Flash devices with normalized rates above 0.9 and simple encoding-decoding.

5) We suggest that non-binary constrained codes are significantly more efficient, rate-wise, compared with binary codes. From [2], the capacity of a binary \( A_4 \)-constrained code (\( x = 1 \)) is 0.8114. From Table IV, we can see that even for \( q = 4 \), a self-clocked QA-LOCO code of length only 20 symbols achieves about 6.4% rate advantage with respect to the aforementioned binary capacity, and at \( x = 2 \) (more ICI mitigation). From the

\[\text{Because of the application, the RLL code used here is not associated with transition-based signaling.}\]
tables, the capacity increase from an $A_1$-constrained code to a $Q_1^{\mathbf{f}}$-constrained (resp., $Q_2^{\mathbf{f}}$-constrained) code is a significant 19.4\% (resp., 16.8\%), which further grows for higher values of $q$.

Because QA-LOCO codes are fixed-length block codes, they prevent error propagation from a codeword into the following ones, and they allow parallel encoding and decoding.

Remark 3: A balanced binary constrained code associated with level-based (NRZ) signaling has the property that the absolute difference between the number of 1’s and 0’s in any stream of its codewords is bounded. Symmetric LOCO codes can be easily balanced with a minimal rate loss as shown in [22]. In the context of $q$-ary constrained codes for Flash, balancing was introduced in [17] as the property that each codeword has uniform distribution for the number of instances of each symbol. Almost-balanced QA-LOCO codes can be designed with less restrictions.

V. ENCODING AND DECODING ALGORITHMS

Now, we introduce the encoding and decoding algorithms of QA-LOCO codes, which are based on their encoding-decoding rule (14) in Theorem 2. The algorithms perform the mapping-demapping between an index and the associated codeword, and thus, they are essential for an enumerative technique to offer simplicity. See [23] for a conceptually connected work in the context of multi-dimensional constellations.

Algorithm 1 is the encoding algorithm of our codes. The algorithm is the reverse operation of (14) in Theorem 2. In particular, we start with the index $g(c)$, and make a series of comparisons to decide the symbols of the codeword $c$ in $Q_{m-1}^{q_{b}}$, $Q_{m}^{q_{b}}$, $Q_{m+1}^{q_{b}}$, and $Q_{m}^{q_{b}}$ from the LMS to the RMS. Each time a symbol $c_i$, $m - 1 \geq i \geq 0$, is decided to be non-zero, its contribution $g_i(c_i)$ is subtracted from the remainder of $g(c)$, which is called residual in Algorithm 1. After the last symbol $c_0$, this residual becomes 0, which is a way to verify the encoding process.

While generating a specific codeword $c$ in the algorithm, the RMS of the previous codeword is defined as $\zeta_0$. Example 5 illustrates how Algorithm 1 works.

Example 5: Consider the CQA-LOCO code $Q_{6,1}^{4,2}(q = 4$, $m = 6$, and $x = 1)$. From Theorem 1, $N_4(-1,1) = 3 - 1$, $N_4(0,1) = 1$, $N_4(1,1) = 4$, $N_4(2,1) = 16$, $N_4(3,1) = 61$, $N_4(4,1) = 232$, $N_4(5,1) = 889$, and $N_4(6,1) = 3409$. Thus, $s^c = \log_{2}3409 = 11$ bits. Now, suppose we want to encode the binary message $b = 11010001110$ via $Q_{6,1}^{4,2}$ using Algorithm 1. From Step 7, $g(c) = \text{decimal}(b) + 1 = 1743$, which is the initial residual from Step 8. The encoding is performed as follows (the loop in Steps 10–39):

1) For $i = 5$, $c_5 \triangleq 0$. Thus, $\gamma_5$ stays 0 (see Steps 11–16), and from Step 17, index $i = 5$. Neither the condition at Step 18 nor the one at Step 20 is satisfied. Thus, the loop starting at Step 24 is entered. Since $N_4(5,1) = 889 < \text{residual} < 2N_4(5,1) = 1778$, $c_5$ is encoded as $L^{-1}(1) = 1$ from Step 26, and residual becomes $1743 - 889 = 854$ from Step 27.

2) For $i = 4$, $c_4 = 1$. Thus, $\gamma_4$ stays 0 (see Steps 11–16), and from Step 17, index $i = 4$. The condition at Step 20 is satisfied since residual $> 3N_4(4,1) = 696$. Thus, $c_4$ is encoded as $\alpha^2$ from Step 21, and residual becomes $854 - 696 = 158$ from Step 22.

3) For $i = 3$, $c_3 = \alpha^2$. Thus, from Steps 12 and 13, $k_3 = 1$. And $\gamma_3 = 1 - 1 + 1 = 1$, and from Step 17, index $i = 1$. The condition at Step 20 is again satisfied since residual $> 9N_4(2,1) = 144$. Thus, $c_3$ is encoded as $\alpha^2$ from Step 21, and residual becomes $158 - 144 = 14$ from Step 22.
The generated codeword is then \( c \). Output:

\[
\begin{align*}
22: & \quad \text{Input stream of binary messages.} \\
21: & \quad \text{Output stream of binary messages.} \\
20: & \quad \text{Ignore the next} \\
19: & \quad \text{Codeword} \\
18: & \quad \text{if} \\
17: & \quad \text{Set} \\
16: & \quad \text{if} \\
15: & \quad \text{Set index} \\
14: & \quad \text{for} \\
13: & \quad \text{Set} \\
12: & \quad \text{for} \\
11: & \quad \text{break. (exit current loop)} \\
10: & \quad \text{end if} \\
9: & \quad \text{end for} \\
8: & \quad \text{if} \\
7: & \quad \text{for} \\
6: & \quad \text{Initialize} \\
5: & \quad \text{for} \\
4: & \quad \text{if} \\
3: & \quad \text{set} \\
2: & \quad \text{if} \\
1: & \quad \text{for} \\
0: & \quad \text{end for} \\
\end{align*}
\]

Algorithm 2 Decoding CQA-LOCO Codes:

1. **Inputs:** Incoming stream of \( q \)-ary CQA-LOCO codewords, in addition to \( q, m, x, \) and \( s^c \).
2. Use (5) and (6) to compute \( N_q(i, x), i \in \{2, 3, \ldots, m-1\} \).
3. For each incoming codeword \( c \) of length \( m \) do:
   4. Initialize \( c(L) \) with 0 and \( c_i \) with 0 for \( i \geq m \).
   5. Initialize \( \gamma_0 \) with 0 for \( i \in \{0, 1, \ldots, m-1\} \).
   6. For \( i \in \{m-1, m-2, \ldots, 0\} \) do (in order):
      7. For \( k_i \in \{1, 2, \ldots, x\} \) do:
         8. If \( c_{i+k} = \alpha^{q-2} \) then:
            9. Set \( \gamma_i = x - k_i + 1 \).
            10. Break. (exit current loop)
      11. End if
   12. End for
   13. Set index \( = i - \gamma_i \).
   14. If \( c_i \neq 0 \) then (same as \( a_i \neq 0 \))
      15. Set \( a_i = L(c_i) \).
      16. Compute \( g(c) = g(c) + a_i(q-1)^\gamma N_q(\text{index}, x) \).
      17. End if
   18. End for
   19. Compute \( b = \text{binary}(g(c) - 1) \), which has length \( s^c \).
20. Ignore the next \( x \) bridging symbols.
21. End for
22. **Output:** Outgoing stream of binary messages.

4) For \( i = 2, c_3 = \alpha^2 \). Thus, from Steps 12 and 13, \( k_2 = 1 \) and \( \gamma_2 = 1 - 1 + 1 = 1 \), and from Step 17, index \( i - \gamma_i = 1 \). Neither the condition at Step 18 nor the one at Step 20 is satisfied. Thus, the loop starting at Step 24 is entered. Since \( 3N_q(1, 1) = 12 < \text{residual} < 6N_q(1, 1) = 24 \), \( c_2 \) is encoded as \( L^{-1}(1) = 1 \) from Step 26, and residual becomes \( 14 - 12 = 2 \) from Step 27.

5) For \( i = 1, c_2 = 1 \), Thus, \( \gamma_1 \) stays 0 (see Steps 11–16), and from Step 17, index \( i = 1 \). The condition at Step 18 is satisfied since residual \( < N_q(1, 1) = 4 \). Thus, \( c_1 \) is encoded as 0 from Step 19, and residual stays 2.

6) For \( i = 0 \), \( c_1 = 0 \). Thus, \( \gamma_0 \) stays 0 (see Steps 11–16), and from Step 17, index \( i = 0 \). Neither the condition at Step 18 nor the one at Step 20 is satisfied. Thus, the loop starting at Step 24 is entered. Since \( 2N_q(0, 1) = 2 < 3N_q(0, 1) = 3 \), \( c_0 \) is encoded as \( L^{-1}(2) = \alpha \) from Step 26, and residual becomes \( 2 - 2 = 0 \) from Step 27.

The generated codeword is then \( c = 10^2 \alpha^2 + 10^0 \alpha \), which is indeed the correct codeword. Bridging is then performed in Steps 32–38.

Algorithm 2 is the decoding algorithm of our codes, and it is a direct implementation of (14). Thus, Example 2 illustrates how Algorithm 2 works.

VI. COMPLEXITY, STORAGE, AND RECONFIGURABILITY

In this section, we discuss the complexity and the storage overhead of the encoding and decoding algorithms of QA-LOCO codes. Additionally, we show how QA-LOCO codes can be reconfigured, contributing to notably better lifecycle management for Flash devices.

In order to reduce complexity, all terms containing multiplications in Algorithms 1 and 2, e.g., \( a_i(q-1)^\gamma N_q(\text{index}, x) \), are not computed at runtime. Instead, these terms are computed offline and stored. This increases the storage overhead, which will be discussed shortly. However, the gain is that the complexity of both algorithms is still mainly governed by the size of the adders that perform the comparisons/subtractions and additions. The adder size is itself the message length \( s^c \). For example, to achieve a rate of 1.8519 input bits per coded symbol using a CQA-LOCO code with \( q = 4 \) and \( x = 1 \), adders of size 1.8519 \times (26 + 1) = 50 bits are needed (see Table III). Another example is, to achieve a rate of 1.8000 input bits per coded symbol using a CQA-LOCO code with \( q = 4 \) and \( x = 2 \), adders of size 1.8000 \times (38 + 2) = 72 bits are needed (see Table IV).

As implied in the previous paragraph, the storage overhead increases as \( q \) increases. In particular, and from Steps 18–31 in Algorithm 1 and Steps 14–17 in Algorithm 2, the storage grows with \( O((q-1)\log_2 q) \), \( q > 2 \), for fixed \( m \). The term \( \log_2 q \) is there because the storage needed for cardinalities only, which are computed offline, grows with \( O(\log_2 q) \) for fixed \( m \). Moreover, from Steps 18–31 in Algorithm 1 (resp., Steps 14–17 in Algorithm 2), the encoding complexity (resp., decoding complexity) grows with \( O((q-1)\log_2 q) \) (resp., \( O(\log_2 q) \)) for fixed \( m \). The term \( \log_2 q \) is there because the adder size grows with \( O(\log_2 q) \) for fixed \( m \). The reason is that the adder size grows linearly with the rate of the code, which grows with \( O(\log_2 q) \) for fixed \( m \). Observe that we use \( (q-1) \) inside the notation \( O(\cdot) \) in order to account for the actual scaling.

However, these orders of growth result in an unfair comparison across different values of \( q \) because they are based on a fixed number of symbols rather than the same amount of coded data in bits. For example, if \( m \) is fixed at 25, these are 25 bits for \( q = 2 \), but equivalent to 50 bits for \( q = 4 \), to 75 bits for \( q = 8 \), and so on. Thus, these orders of growth should be divided by \( \log_2 q \) for a fair comparison, which results in \( O((q-1)\log_2 q) \) for storage, \( O((q-1)\log_2 q) \) for encoding complexity, and \( O(1) \) for decoding complexity. Thus, the storage and complexity of QA-LOCO encoding and decoding with \( q > 2 \) are still manageable, and are less than those of other enumerative techniques. In general, any \( q \)-ary signal processing technique has complexity that increases as \( q \) increases. One useful analogy to make is with the complexity growth of non-binary low-density parity-check decoding, which has \( O(q(\log_2 q)) \) [24] that goes down to \( O(q) \) for fixed length in bits. The QA-LOCO order of storage growth is similar to that order for fixed \( x \), the QA-LOCO order of encoding-complexity growth is the

[8]The QA-LOCO encoding-decoding complexity discussed here is the worst-case complexity. Moreover, the storage overhead (for fixed \( x \)) and the encoding-decoding complexity of a QA-LOCO code increase as \( m \) increases the same way for any fixed \( q \). For complexity, this way is linear on the level of arithmetic operations per codeword.

We can also choose to fix the message length in bits instead of fixing the amount of coded data. Note that while QA-LOCO codes with higher values of \( q \) have higher rates, the effect of this on the orders of growth is minor.
same as that order, and the QA-LOCO order of decoding-complexity growth is even much better than that order.

Remark 4: We are working on ideas to further reduce the storage overhead and the encoding-decoding complexity for QA-LOCO codes. Regarding the encoding complexity for example, one idea is to change the for loop in Steps 24–30 in Algorithm 1 such that it performs a binary search, reducing the order of encoding-complexity growth to $O(\log_2 q)$ for fixed length in bits. Another idea that is more general is to divide the $q$ symbols, for $q \geq 16$, into 8 groups and apply a QA-LOCO code with $q = 8$. While this idea results in a small rate loss, it achieves notable storage and complexity savings.

A Flash device with $q$ levels per cell has $\log_2 q$ pages. Each word line in a Flash memory block has these $\log_2 q$ pages in its cells. In general, the Flash industry prefers to process different pages independently in order to increase access speed. One idea to preserve high access speed is to apply the QA-LOCO code only on the parity part of the component EC low-density parity-check (LDPC) code as we did in [22] for magnetic recording (MR) systems. In particular, the idea is to reserve few word lines in each Flash block for parity. We group the parity bits of each set of $\log_2 q$ binary LDPC codewords that have their input information bits to be written over the $\log_2 q$ different Flash pages of the same word line; one codeword per page. Then, we associate them with the parity bits of codewords to be written over the pages of other word lines, and we encode them via a QA-LOCO code into symbols over $GF(q)$ before writing them on the parity word lines; one symbol per cell. The $Q^g_k$ constraint should be satisfied over all bit lines across the parity word lines. While reading, we read the parity word lines first. Next, the parity bits are decoded via the QA-LOCO decoder and associated with their respective codewords. Then, the LDPC decoder operates independently on the $\log_2 q$ pages to retrieve the $\log_2 q$ codewords for each word line. High performance LDPC codes for Flash can be designed as in [15], [16], and [25].

The fact that the encoding and decoding of QA-LOCO codes are performed through simple adders enables reconfigurability. All that is needed to reconfigure a QA-LOCO code, i.e., change the code parameters such that more (or even different) constraints are supported, is to change the cardinalities that are inputs to the adders at both the encoding and decoding sides such that the encoding-decoding rule in (14) supports the new constraints. As the Flash device ages, charges propagate during programming with higher rates and reach further non-adjacent cells. Thus, while QA-LOCO codes with $x = 1$ are sufficient when the device is fresh, reconfiguring to QA-LOCO codes with $x > 1$, i.e., forbidding more patterns, is needed such that the device keeps functioning reliably late in its lifetime.

Aided by machine learning, errors before the LDPC decoder can be collected to identify the set of error-prone patterns that should be forbidden at different stages of the Flash device lifetime. Once this set is found to be bigger than the currently supported set by the QA-LOCO code, we propose to respond via reconfiguring the QA-LOCO code to support the new set as illustrated in the previous paragraph. Therefore, machine learning and reconfigurable constrained codes, along with high performance EC codes, can help increase the lifetime of modern Flash devices significantly, and therefore support the evolution of QLC and PLC Flash memories.

Flexible techniques to design reconfigurable NAND Flash memories [26] have the potential to achieve a great compromise between the higher storage density offered by M/T/Q/P-MLC Flash memories on the one hand, and the higher speed and endurance offered by SLC Flash memories on the other hand. Observe that here we are talking about the reconfigurability of the Flash storage itself. In particular, the fraction of cells programmed as M/T/Q/P-LCs and that of cells programmed as SLCs can be updated at runtime depending on the application.

Since our encoding-decoding rule is just a summation, QA-LOCO codes perfectly support such reconfigurable Flash memories. Just the right cardinalities need to be plugged in as inputs to the adders, and the new QA-LOCO codes will be ready for the new programming.

VII. CONCLUSION

We introduced capacity-achieving $q$-ary asymmetric LOCO codes (QA-LOCO codes) for Flash devices with any number, $q$, of levels per cell. We partitioned the codewords of a QA-LOCO code into groups, which we used to recursively compute the cardinality. We devised an encoding-decoding rule for QA-LOCO codes to map-demap from index to codeword and vice versa, which is the key result behind the simple encoding and decoding of these codes. We introduced the achievable rates of QA-LOCO codes, and showed that they need less than 5% redundancy to protect the device. For QLC and PLC devices, we demonstrated that ICI mitigation almost comes for free with respect to redundancy. We presented the encoding and decoding algorithms, and provided an analysis for the complexity and storage growth with $q$. We suggest that machine learning and reconfigurable QA-LOCO codes (with EC codes) can significantly increase the lifetime of modern Flash devices. Future work includes designing two-dimensional LOCO codes for two-dimensional MR devices in addition to theoretically analyzing ideas to incorporate machine learning in data storage.

ACKNOWLEDGMENT

The authors would like to thank the Associate Editor Prof. Anxiao Jiang for his expeditious handling of the article and his constructive feedback. The authors would also like to thank the anonymous reviewers for their valuable and helpful comments (this extends to the ISIT TPC Member and reviewers as well).

REFERENCES

[1] A. Hareedy, B. Dabak, and R. Calderbank, “Q-ary asymmetric LOCO codes: Constrained codes supporting flash evolution,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Los Angeles, CA, USA, Jun. 2020, pp. 688–693.
[2] A. Hareedy and R. Calderbank, “Asymmetric LOCO codes: Constrained codes for flash memories,” in Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton), Monticello, IL, USA, Sep. 2019, pp. 124–131.
Ahmed Hareedy (Member, IEEE) is a Postdoctoral Associate with the Electrical and Computer Engineering Department at Duke University. He is interested in questions in coding/information theory that are fundamental to opportunities created by the current, unparalleled access to data and computing. He received the Bachelor and M.S. degrees in Electronics and Communications Engineering from Cairo University in 2006 and 2011, respectively. He received the Ph.D. degree in Electrical and Computer Engineering from the University of California, Los Angeles (UCLA) in 2018. He worked with Mentor Graphics Corporation between 2006 and 2014. He worked as an Error-Correction Coding Architect with Intel Corporation in the summers of 2015 and 2017.

Dr. Hareedy won the 2018–2019 Distinguished Ph.D. Dissertation Award in Signals and Systems from the Electrical and Computer Engineering Department at UCLA. He is a recipient of the Best Paper Award from the 2015 IEEE Global Communications Conference (GLOBECOM), Selected Areas in Communications, Data Storage Track. He won the 2017–2018 Dissertation Year Fellowship (DYF) at UCLA. He won the 2016–2017 Electrical Engineering Henry Samueli Excellence in Teaching Award for teaching Probability and Statistics at UCLA. He is a recipient of the Memorable Paper Award from the 2018 Non-Volatile Memories Workshop (NVMM) in the area of devices, coding, and information theory. He is a recipient of the 2018–2019 Best Student Paper Award from the IEEE Data Storage Technical Committee (DSTC).

Bezya Dabak received the B.S. degree in Electrical and Electronics Engineering from Bilkent University, Turkey in 2019. She is a Ph.D. student in Electrical and Computer Engineering Department at Duke University. Her research interests include questions in coding/information theory with data storage applications.

Robert Calderbank (Fellow, IEEE) received the B.S. degree in 1975 from Warwick University, England, the M.S. degree in 1976 from Oxford University, England, and the Ph.D. degree in 1980 from the California Institute of Technology, all in Mathematics.

Dr. Calderbank is a Professor of Electrical and Computer Engineering at Duke University where he directs the Rhodes Information Initiative at Duke (iDi). Prior to joining Duke in 2010, Dr. Calderbank was a Professor of Electrical Engineering and Mathematics at Princeton University where he directed the Program in Applied and Computational Mathematics. Prior to joining Princeton in 2004, he was the Vice President for Research at AT&T Labs, responsible for directing the first industrial research lab in the world where the primary focus is data at scale. At the start of his career at Bell Labs, innovations by Dr. Calderbank were incorporated in a progression of voiceband modem standards that moved communications practice close to the Shannon limit. Together with Peter Shor and colleagues at AT&T Labs he developed the mathematical framework for quantum error correction. He is a co-inventor of space-time codes for wireless communication, where correlation of signals across different transmit antennas is the key to reliable transmission.

Dr. Calderbank served as the Editor in Chief of the IEEE TRANSACTIONS ON INFORMATION THEORY from 1995 to 1998, and as an Associate Editor for Coding Techniques from 1986 to 1989. He was a member of the Board of Governors of the IEEE Information Theory Society from 1991 to 1996 and from 2006 to 2008. Dr. Calderbank was honored by the IEEE Information Theory Prize Paper Award in 1995 for his work on the Z4 linearity of Kerdock and Preparata Codes (joint with A.R. Hammons Jr., P.V. Kumar, N.J.A. Sloane, and P. Sole), and again in 1999 for the invention of space-time codes (joint with V. Tarokh and N. Seshadhri). He has received the 2006 IEEE Donald G. Fink Prize Paper Award, the IEEE Millennium Medal, the 2013 IEEE Richard W. Hamming Medal, and the 2015 Shannon Award. He was elected to the US National Academy of Engineering in 2005.