Rank Modulation in Flash Memories

Course Project, EE 605

Saketika Chekuri[1] Suraj Samaga[2]

[1]190070054
Electrical Engineering, IIT Bombay

[2]190020114
Electrical Engineering, IIT Bombay

Error Correcting Codes, Autumn 2021

Table of Contents

Introduction

Flash memory is a non-volatile memory that is both electrically programmable and erasable.

Introduction

Flash memory is a non-volatile memory that is both electrically programmable and erasable. Historically, it comprises of blocks of cells, where each cell is in one of two states. However, multi-level cells, where each cell stores one of q levels are being actively developed.

Introduction

Flash memory is a non-volatile memory that is both electrically programmable and erasable. Historically, it comprises of blocks of cells, where each cell is in one of two states. However, multi-level cells, where each cell stores one of q levels are being actively developed.

Device electronics tells us that while adding charge to a single cell is a fast and simple operation, removing charge from a single cell is very difficult.

Introduction

Flash memory is a non-volatile memory that is both electrically programmable and erasable. Historically, it comprises of blocks of cells, where each cell is in one of two states. However, multi-level cells, where each cell stores one of q levels are being actively developed.

Device electronics tells us that while adding charge to a single cell is a fast and simple operation, removing charge from a single cell is very difficult. This calls for accurate programming schemes that cautiously approach target charge levels from below.

Introduction

Additionally, the move to multi-level flash cells aggravates device reliability.

Introduction

Additionally, the move to multi-level flash cells aggravates device reliability. For example, sufficient drift in device threshold levels may cause programming and reading errors.

Introduction

Additionally, the move to multi-level flash cells aggravates device reliability. For example, sufficient drift in device threshold levels may cause programming and reading errors.

Problem

Designing codes where k bits are stored using a block of n cells with q levels each, with the additional requirement of addressing the challenges above.

Introduction

Additionally, the move to multi-level flash cells aggravates device reliability. For example, sufficient drift in device threshold levels may cause programming and reading errors.

Problem

Designing codes where k bits are stored using a block of n cells with q levels each, with the additional requirement of addressing the challenges above.

One such scheme is Rank Modulation.

Rank Modulation

- An ordered set of n cells stores the information in the permutation induced by the charge levels of the cells

Rank Modulation

- An ordered set of n cells stores the information in the permutation induced by the charge levels of the cells
- The only allowed programming operation is a ‘push-to-the-top’ operation, raising the charge level of one of the cells to above the current highest one

Rank Modulation

- An ordered set of n cells stores the information in the permutation induced by the charge levels of the cells
- The only allowed programming operation is a 'push-to-the-top' operation, raising the charge level of one of the cells to above the current highest one
- Additionally, 'block-deflation', decrease of all the charge levels in a block by a constant amount smaller than the lowest, maintaining relative values is allowed

Rank Modulation - Gray Codes

We use Gray Codes to implement Rank Modulation.

Rank Modulation - Gray Codes

We use Gray Codes to implement Rank Modulation.

Definition 1 (Gray Codes)

Ordered set of distinct states for which every state s_i is followed by a state s_{i+1} such that $s_{i+1} = t(s_i)$

Rank Modulation - Gray Codes

We use Gray Codes to implement Rank Modulation.

Definition 1 (Gray Codes)

Ordered set of distinct states for which every state s_i is followed by a state s_{i+1} such that $s_{i+1} = t(s_i)$ where $t \in T$ is a transition function from a predetermined set T defining the Gray Code

Rank Modulation - Gray Codes

We use Gray Codes to implement Rank Modulation.

Definition 1 (Gray Codes)

Ordered set of distinct states for which every state s_i is followed by a state s_{i+1} such that $s_{i+1} = t(s_i)$ where $t \in T$ is a transition function from a predetermined set T defining the Gray Code

The traversal of states by the Gray Code is mapped to the increase of cell level in the multi-level flash cell.

Table of Contents

Notation

- We consider each relative ordering of the n cells as a state, belonging to state space S.

Notation

- We consider each relative ordering of the n cells as a state, belonging to state space S.
- Let $[n]$ denote the set $\{1, 2, \ldots n\}$. An ordered set of n memory cells labelled $1, 2 \ldots n$ which distinct charge levels induce permutations of $[n]$ by representing the cell names as vectors $[a_1, a_2, \ldots a_n]$ in descending order of charge levels.
- State space would be all possible permutations on $[n]$.

Notation

- For the rank modulation scheme, we only consider push-to-the-top operations.

Notation

- For the rank modulation scheme, we only consider push-to-the-top operations.
- Let t_i represent the transition that pushes the ith highest cell to the highest charge level.
  $t_i([a_1, a_2, \ldots, \boldsymbol{a_i}, a_{i+1}, \ldots a_n]) = [\boldsymbol{a_i}, a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n]$

Notation

- For the rank modulation scheme, we only consider push-to-the-top operations.
- Let t_i represent the transition that pushes the ith highest cell to the highest charge level.
  $t_i([a_1, a_2, \ldots, \boldsymbol{a_i}, a_{i+1}, \ldots a_n]) = [\boldsymbol{a_i}, a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n]$
- Let T be the set of all push-to-the-top transitions. That is, $s_{i+1} = t(s_i)$ for some $t \in T, s_i, s_{i+1} \in S$.

Definitions

- The code we consider, in which state space is all permutations on $[n]$ and the set of our transitions is all possible push-to-the-top functions is called a "length-n rank modulation Gray code" (**n-RMGC**).

Definitions

- The code we consider, in which state space is all permutations on $[n]$ and the set of our transitions is all possible push-to-the-top functions is called a "length-n rank modulation Gray code" (**n-RMGC**).
- If the code is such that $s_1 = t(s_m)$ for some $t \in T$, the code is **cyclic**.
- If the code spans all of S, it is considered **complete**.

Example

Example 2

A 3-RMGC is given below:

$$\begin{array}{cccccc} 1 & 2 & 3 & 1 & 3 & 2 \\ 2 & 1 & 2 & 3 & 1 & 3 \\ 3 & 3 & 1 & 2 & 2 & 1 \end{array}$$

Here, the permutations are the different columns.

Example

Example 2

A 3-RMGC is given below:

$$\begin{array}{cccccc} 1 & 2 & 3 & 1 & 3 & 2 \\ 2 & 1 & 2 & 3 & 1 & 3 \\ 3 & 3 & 1 & 2 & 2 & 1 \end{array}$$

Here, the permutations are the different columns.This code is defined by the transitions $t_2, t_3, t_3, t_2, t_3, t_3$. Note that the last t_3 transition gives back $1, 2, 3$, making the code cyclic.

Construction of n-RMGC

- Construct cyclic and complete RMGCs via recursion.
- Recursion basis is 2-RMGC: $[1, 2], [2, 1]$

Construction of n-RMGC

- Construct cyclic and complete RMGCs via recursion.
- Recursion basis is 2-RMGC: $[1, 2], [2, 1]$
- Assume cyclic and complete $(n-1)$-RMGC already available called C_{n-1} defined by transitions $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ with $\boldsymbol{t_{i_{(n-1)!}} = t_2}$

Construction of n-RMGC

- Construct cyclic and complete RMGCs via recursion.
- Recursion basis is 2-RMGC: $[1, 2], [2, 1]$
- Assume cyclic and complete $(n-1)$-RMGC already available called C_{n-1} defined by transitions $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ with $\boldsymbol{t_{i_{(n-1)!}} = t_2}$
- The $t_{i_{(n-1)!}} = t_2$ condition is not very restrictive, since it only necessitates a t_2 transition somewhere in the transition sequence, and we can always rotate the sequence to make t_2 the last one.

Construction of n-RMGC

- Construct cyclic and complete RMGCs via recursion.
- Recursion basis is 2-RMGC: $[1, 2], [2, 1]$
- Assume cyclic and complete $(n-1)$-RMGC already available called C_{n-1} defined by transitions $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ with $\boldsymbol{t_{i_{(n-1)!}} = t_2}$
- The $t_{i_{(n-1)!}} = t_2$ condition is not very restrictive, since it only necessitates a t_2 transition somewhere in the transition sequence, and we can always rotate the sequence to make t_2 the last one.
- We further assume t_2 appears at least twice.

Construction

- Without loss of generality, assume the first permutation is $[1, 2, \ldots, n]$.
- Using $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ from C_{n-1} (and thus effectively keeping the last element fixed), we get the first *block* of $(n-1)!$ elements of our construction.

Construction

- Without loss of generality, assume the first permutation is $[1, 2, \ldots, n]$.
- Using $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ from C_{n-1} (and thus effectively keeping the last element fixed), we get the first *block* of $(n-1)!$ elements of our construction.
- Since $t_{i_{(n-1)!}} = t_2$, the last element of the first block is $[2, 1, 3, 4, \ldots, n-1, n]$

Construction

- Without loss of generality, assume the first permutation is $[1, 2, \ldots, n]$.
- Using $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ from C_{n-1} (and thus effectively keeping the last element fixed), we get the first *block* of $(n-1)!$ elements of our construction.
- Since $t_{i_{(n-1)!}} = t_2$, the last element of the first block is $[2, 1, 3, 4, \ldots, n-1, n]$
- For the first element of the second block, we first use t_n, and use $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ again.

Construction

- Without loss of generality, assume the first permutation is $[1, 2, \ldots, n]$.
- Using $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ from C_{n-1} (and thus effectively keeping the last element fixed), we get the first *block* of $(n-1)!$ elements of our construction.
- Since $t_{i_{(n-1)!}} = t_2$, the last element of the first block is $[2, 1, 3, 4, \ldots, n-1, n]$
- For the first element of the second block, we first use t_n, and use $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ again.
- This time, $(n-1)$ would be the fixed last element while all the other elements create $(n-1)!$ more permutations.

Construction

- Without loss of generality, assume the first permutation is $[1, 2, \ldots, n]$.
- Using $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ from C_{n-1} (and thus effectively keeping the last element fixed), we get the first *block* of $(n-1)!$ elements of our construction.
- Since $t_{i_{(n-1)!}} = t_2$, the last element of the first block is $[2, 1, 3, 4, \ldots, n-1, n]$
- For the first element of the second block, we first use t_n, and use $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ again.
- This time, $(n-1)$ would be the fixed last element while all the other elements create $(n-1)!$ more permutations.
- Repeat this process $(n-1)$ times to get $(n-1)!$ permutations each time.

Construction

Lemma 3

Second element in first permutation to every block is 2. It follows that first element of last permutation in every block is also 2.

Proof.

We use the transitions $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!-1}}$ in this particular order. If we use $t_{i_{(n-1)!}} = t_2$, we get the first permutation of the block back, which has 2 in the second position. Therefore, after $t_{i_{(n-1)!-1}}$, we get the last permutation of the block in which 2 must have been the first element. $\square$

Construction

Lemma 4

In any block, last element of all permutations is always constant, and the sequence of last elements starting from $[1, 2, \ldots, n]$ *is always* $n, n-1, \ldots, 4, 3, 1$. *2 is never the last element.*

Proof.

- The first claim is justified since we use transitions that define C_{n-1} which only operate on the first $(n-1)$ elements.
- 2 is the first element for the last permutation of each block from Lemma 3, so it can't simultaneously the last element as well.

$\square$

Construction

- Let the set of $(n-1)$ blocks we constructed be C'.
- From the lemmas above, we can see that C' is not complete yet since it doesn't have any permutations ending in 2.

Construction

- Let the set of $(n-1)$ blocks we constructed be C'.
- From the lemmas above, we can see that C' is not complete yet since it doesn't have any permutations ending in 2.
- We provide an alternate construction to build C'', the set of all permutations with 2 as last element.

Construction of C''

- Start by rotating $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ so that the last transition is t_{n-1}.

Construction of C''

- Start by rotating $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ so that the last transition is t_{n-1}. Let this rotated sequence be $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_{(n-1)!}}$ such that $\tau_{i_{(n-1)!}} = t_{n-1}$

Construction of C''

- Start by rotating $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ so that the last transition is t_{n-1}. Let this rotated sequence be $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_{(n-1)!}}$ such that $\tau_{i_{(n-1)!}} = t_{n-1}$
- We start with the first permutation in the block as $[a_1, a_2, \ldots, a_{n-1}, 2]$. Set the permutations of C'' as those formed by $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_{(n-1)!-1}}$ starting from this first permutation.

Construction of C''

- Start by rotating $t_{i_1}, t_{i_2}, \ldots, t_{i_{(n-1)!}}$ so that the last transition is t_{n-1}. Let this rotated sequence be $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_{(n-1)!}}$ such that $\tau_{i_{(n-1)!}} = t_{n-1}$
- We start with the first permutation in the block as $[a_1, a_2, \ldots, a_{n-1}, 2]$. Set the permutations of C'' as those formed by $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_{(n-1)!-1}}$ starting from this first permutation.
- The last permutation of C'' will therefore be $[a_2, a_3, \ldots, a_{n-1}, a_1, 2]$

Construction

- In C', we look for a transition of the form $[a_2, a_3, \ldots, a_{n-1}, 2, a_1] \to [2, a_2, \ldots, n-1, a_1]$

Construction

- In C', we look for a transition of the form $[a_2, a_3, \ldots, a_{n-1}, 2, a_1] \to [2, a_2, \ldots, n-1, a_1]$
- Such a transition would surely exist; while C' doesn't have any permutations in which 2 is the last element, it does have permutations in which 2 is the penultimate element and others in which 2 is the first element. Since C' is cyclic, such a transition would surely exist.

Construction

- In C', we look for a transition of the form $[a_2, a_3, \ldots, a_{n-1}, 2, a_1] \rightarrow [2, a_2, \ldots, n-1, a_1]$
- Such a transition would surely exist; while C' doesn't have any permutations in which 2 is the last element, it does have permutations in which 2 is the penultimate element and others in which 2 is the first element. Since C' is cyclic, such a transition would surely exist.
- Whenever such a transition occurs (note that there can be multiple such occurrences, so we can choose any one), we split C' and insert C'' to create a complete RMGC as follows:

$$\begin{array}{c} a_2\\ a_3\\ \vdots\\ a_{n-1}\\ 2\\ a_1 \end{array} \underset{t_n}{\longrightarrow} \boxed{\begin{array}{ccc} a_1 & & a_2\\ a_2 & & a_3\\ \vdots & \rightsquigarrow & \vdots\\ a_{n-2} & & a_{n-1}\\ a_{n-1} & & a_1\\ 2 & & 2 \end{array}} \underset{t_n}{\longrightarrow} \begin{array}{c} 2\\ a_2\\ \vdots\\ a_{n-2}\\ a_{n-1}\\ a_1 \end{array}$$

Summary

Theorem 5

For every integer $n \geq 2$, there exists a cyclic and complete n-RMGC.

Example

Example 6

As considered in Example 2, we start the 3-RMGC we obtained. However, this had the transition sequence $t_2, t_3, t_3, t_2, t_3, t_3$, but we want it to end in t_2

Example

Example 6

As considered in Example 2, we start the 3-RMGC we obtained. However, this had the transition sequence $t_2, t_3, t_3, t_2, t_3, t_3$, but we want it to end in t_2. So, we rotate the sequence to $t_3, t_3, t_2, t_3, t_3, t_2$ and obtain the following C' starting with $[1, 2, 3, 4]$:

$$\boxed{\begin{array}{cccccc} 1 & 3 & 2 & 3 & 1 & 2 \\ 2 & 1 & 3 & 2 & 3 & 1 \\ 3 & 2 & 1 & 1 & 2 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{array}} \boxed{\begin{array}{cccccc} 4 & 1 & 2 & 1 & 4 & 2 \\ 2 & 4 & 1 & 2 & 1 & 4 \\ 1 & 2 & 4 & 4 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{array}} \boxed{\begin{array}{cccccc} 3 & 4 & 2 & 4 & 3 & 2 \\ 2 & 3 & 4 & 2 & 4 & 3 \\ 4 & 2 & 3 & 3 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}}$$

Example

To construct C'', we look for a transition in which 2 goes from the penultimate position to the top. Multiple such transitions exist, so we choose $[1, 3, 2, 4] \rightarrow [2, 1, 3, 4]$. Inserting C'' between C', we get:

$$\begin{matrix}
1 & 3 & 2 & 3 & 1 & 4 & 1 & 3 & 4 & 3 & 1 & 2 & 4 & 1 & 2 & 1 & 4 & 2 & 3 & 4 & 2 & 4 & 3 & 2 \\
2 & 1 & 3 & 2 & 3 & 1 & 4 & 1 & 3 & 4 & 3 & 1 & 2 & 4 & 1 & 2 & 1 & 4 & 2 & 3 & 4 & 2 & 4 & 3 \\
3 & 2 & 1 & 1 & 2 & 3 & 3 & 4 & 1 & 1 & 4 & 3 & 1 & 2 & 4 & 4 & 2 & 1 & 4 & 2 & 3 & 3 & 2 & 4 \\
4 & 4 & 4 & 4 & 4 & 2 & 2 & 2 & 2 & 2 & 2 & 4 & 3 & 3 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 1 & 1 & 1
\end{matrix}$$

Table of Contents

Definitions

While the construction seen in the previous section is straightforward and intuitive, it suffers from a serious drawback.

Definitions

While the construction seen in the previous section is straightforward and intuitive, it suffers from a serious drawback.While the top-charged cells are changed regularly while permuting within a block, the bottom cell remains unchanged and is only activated at the end of each block.

Definitions

While the construction seen in the previous section is straightforward and intuitive, it suffers from a serious drawback.While the top-charged cells are changed regularly while permuting within a block, the bottom cell remains unchanged and is only activated at the end of each block.This leads to a large gap between the least charged and most charged cells, which is undesirable.

Definitions

Definition 7

We define $c_i : \mathbb{N} \mapsto \mathbb{N}$ where $c_i(p)$ is the charge level of the i^{th} cell after the p^{th} programming cycle.

Definitions

Definition 7

We define $c_i : \mathbb{N} \mapsto \mathbb{N}$ where $c_i(p)$ is the charge level of the i^{th} cell after the p^{th} programming cycle.

- Suppose we use transition t_j in the p^{th} programming cycle, and the i^{th} cell is at that time, j^{th} from the top

Definitions

Definition 7

We define $c_i : \mathbb{N} \mapsto \mathbb{N}$ where $c_i(p)$ is the charge level of the i^{th} cell after the p^{th} programming cycle.

- Suppose we use transition t_j in the p^{th} programming cycle, and the i^{th} cell is at that time, j^{th} from the top
- $c_i(p) > \max_k c_k(p-1)$

Definitions

Definition 7

We define $c_i : \mathbb{N} \mapsto \mathbb{N}$ where $c_i(p)$ is the charge level of the i^{th} cell after the p^{th} programming cycle.

- Suppose we use transition t_j in the p^{th} programming cycle, and the i^{th} cell is at that time, j^{th} from the top
- $c_i(p) > \max_k c_k(p-1)$
- for $k \neq i, c_k(p) = c_k(p-1)$

Definitions

Definition 7

We define $c_i : \mathbb{N} \mapsto \mathbb{N}$ where $c_i(p)$ is the charge level of the i^{th} cell after the p^{th} programming cycle.

- Suppose we use transition t_j in the p^{th} programming cycle, and the i^{th} cell is at that time, j^{th} from the top
- $c_i(p) > \max_k c_k(p-1)$
- for $k \neq i, c_k(p) = c_k(p-1)$
- In an optimal setting with no overshoots, or 'gaps', $c_i(p) = \max_k c_k(p-1) + 1$

Definitions

- Assuming the i^{th} cell was affected (moved) in the p^{th} transition, 'jump' is defined as $c_i(p) - c_i(p-1)$

Definitions

- Assuming the i^{th} cell was affected (moved) in the p^{th} transition, 'jump' is defined as $c_i(p) - c_i(p-1)$
- **Jump Cost** of an n-RMGC is defined as the maximum jump over all transitions of the code

Definitions

- Assuming the i^{th} cell was affected (moved) in the p^{th} transition, 'jump' is defined as $c_i(p) - c_i(p-1)$
- **Jump Cost** of an n-RMGC is defined as the maximum jump over all transitions of the code
- An n-RMGC is said to be optimal if it's jump cost is not larger than any other n-RMGC

Definitions

- Assuming the i^{th} cell was affected (moved) in the p^{th} transition, 'jump' is defined as $c_i(p) - c_i(p-1)$
- **Jump Cost** of an n-RMGC is defined as the maximum jump over all transitions of the code
- An n-RMGC is said to be optimal if it's jump cost is not larger than any other n-RMGC
- In the above definition we assume that there are no degenerate cells, i.e. every cell level is raised at-least once

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ *without any degenerate cells, the jump cost is at least* $n+1$

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ without any degenerate cells, the jump cost is at least $n + 1$

Proof.

- The transition t_n would be the worst-case one to consider while evaluating jump cost

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ without any degenerate cells, the jump cost is at least $n + 1$

Proof.

- The transition t_n would be the worst-case one to consider while evaluating jump cost
- Such a jump has magnitude of at-least n (when all charge levels are consecutive) and we must use it at-least n times to cover all states

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ without any degenerate cells, the jump cost is at least $n + 1$

Proof.

- The transition t_n would be the worst-case one to consider while evaluating jump cost
- Such a jump has magnitude of at-least n (when all charge levels are consecutive) and we must use it at-least n times to cover all states
- However we cannot apply t_n n times consecutively, as that would lead us to the first permutation after just n steps

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ without any degenerate cells, the jump cost is at least $n + 1$

Proof.

- The transition t_n would be the worst-case one to consider while evaluating jump cost
- Such a jump has magnitude of at-least n (when all charge levels are consecutive) and we must use it at-least n times to cover all states
- However we cannot apply t_n n times consecutively, as that would lead us to the first permutation after just n steps
- Hence our set of transitions must at-least one other transition $t_i, i \neq n$

Definitions

Lemma 8

For any optimal n-RMGC, $n \geq 3$ without any degenerate cells, the jump cost is at least $n + 1$

Proof.

- The transition t_n would be the worst-case one to consider while evaluating jump cost
- Such a jump has magnitude of at-least n (when all charge levels are consecutive) and we must use it at-least n times to cover all states
- However we cannot apply t_n n times consecutively, as that would lead us to the first permutation after just n steps
- Hence our set of transitions must at-least one other transition $t_i, i \neq n$
- This would cause a 'gap' in the charge levels, and the first t_n to be used after it will have a jump of magnitude at-least $n + 1$

Balanced n-RMGC

- An n-RMGC that achieves the optimal jump cost of $n+1$ is called a *Balanced n-RMGC*

Balanced n-RMGC

- An n-RMGC that achieves the optimal jump cost of $n+1$ is called a *Balanced n-RMGC*
- We now try to come up with a construction that converts any $(n-1)$-RMGC that is cyclic(complete) into a balanced n-RMGC that is cyclic(complete)

Balanced n-RMGC

- An n-RMGC that achieves the optimal jump cost of $n+1$ is called a *Balanced n-RMGC*
- We now try to come up with a construction that converts any $(n-1)$-RMGC that is cyclic(complete) into a balanced n-RMGC that is cyclic(complete)
- We observe that the transition t_n is the only one that does not introduce gaps between charge levels.

Balanced n-RMGC

- An n-RMGC that achieves the optimal jump cost of $n+1$ is called a *Balanced n-RMGC*
- We now try to come up with a construction that converts any $(n-1)$-RMGC that is cyclic(complete) into a balanced n-RMGC that is cyclic(complete)
- We observe that the transition t_n is the only one that does not introduce gaps between charge levels.
- Hence we base our construction around t_n and try to use it as often as possible

Balanced n-RMGC Construction

Theorem 9

Given a cyclic and complete $(n-1)$-RMGC C_{n-1} defined by the transitions $t_{i_1}, \ldots, t_{i_{(n-1)!}}$, the following transitions define an n-RMGC that is cyclic, complete and balanced

$$t_{j_k} = \begin{cases} t_{n-i_{\lceil k/n \rceil}+1} & k \equiv 1 (mod n) \\ t_n & otherwise \end{cases}$$

for all $k \in \{1, \ldots, n\}$

Balanced n-RMGC Construction

Proof.

- Consider the abstract transition $\bar{t}_i, 2 \leq i \leq n$ that pushes the i^{th} element from the bottom to the bottom of a permutation

Balanced n-RMGC Construction

Proof.

- Consider the abstract transition $\bar{t}_i, 2 \leq i \leq n$ that pushes the i^{th} element from the bottom to the bottom of a permutation
- $\bar{t}_i([a_1, a_2, \ldots, \boldsymbol{a_{n-i+1}}, a_{i+1}, \ldots a_n]) = [a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n, \boldsymbol{a_{n-i+1}}]$

Balanced n-RMGC Construction

Proof.

- Consider the abstract transition $\bar{t}_i, 2 \leq i \leq n$ that pushes the i^{th} element from the bottom to the bottom of a permutation
- $\bar{t}_i([a_1, a_2, \ldots, \boldsymbol{a_{n-i+1}}, a_{i+1}, \ldots a_n]) = [a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n, \boldsymbol{a_{n-i+1}}]$
- Since C_{n-1} is cyclic and complete, using $\{\bar{t}_i\}_{(n-1)!}$ we can traverse all permutations of $[n-1]$

Balanced n-RMGC Construction

Proof.

- Consider the abstract transition $\bar{t}_i, 2 \leq i \leq n$ that pushes the i^{th} element from the bottom to the bottom of a permutation
- $\bar{t}_i([a_1, a_2, \ldots, \boldsymbol{a_{n-i+1}}, a_{i+1}, \ldots a_n]) = [a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n, \boldsymbol{a_{n-i+1}}]$
- Since C_{n-1} is cyclic and complete, using $\{\bar{t}_i\}_{(n-1)!}$ we can traverse all permutations of $[n-1]$
- Using the above with $[n]$ will thus generate all $(n-1)!$ permutations of [n] with first element fixed

Balanced n-RMGC Construction

Proof.

- Consider the abstract transition $\bar{t}_i, 2 \leq i \leq n$ that pushes the i^{th} element from the bottom to the bottom of a permutation
- $\bar{t}_i([a_1, a_2, \ldots, \boldsymbol{a_{n-i+1}}, a_{i+1}, \ldots a_n]) = [a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots a_n, \boldsymbol{a_{n-i+1}}]$
- Since C_{n-1} is cyclic and complete, using $\{\bar{t}_i\}_{(n-1)!}$ we can traverse all permutations of $[n-1]$
- Using the above with $[n]$ will thus generate all $(n-1)!$ permutations of [n] with first element fixed
- It is not hard to see that

$$\bar{t}_i(\sigma) = (t_n \circ \ldots (n - 1 times) \circ t_n \circ t_{n-i+1})(\sigma), \forall \sigma \in S_n$$

Balanced n-RMGC Construction

contd.

$$\underbrace{t_{n-i_1+1}, \overbrace{t_n, \dots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_1}}}, \dots, \underbrace{t_{n-i_{(n-1)!}+1}, \overbrace{t_n, \dots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_{(n-1)!}}}}.$$

Figure: Transition sequence

- From the above two points, the sequence of transitions to be applied to $[n]$ to generate a balanced n-RMGC is as shown in 1

Balanced n-RMGC Construction

contd.

$$\underbrace{t_{n-i_1+1},\overbrace{t_n,\ldots,t_n}^{n-1\text{ times}}}_{\overrightarrow{t_{i_1}}},\ldots,\underbrace{t_{n-i_{(n-1)!}+1},\overbrace{t_n,\ldots,t_n}^{n-1\text{ times}}}_{\overrightarrow{t_{i_{(n-1)!}}}}.$$

Figure: Transition sequence

- From the above two points, the sequence of transitions to be applied to $[n]$ to generate a balanced n-RMGC is as shown in 1
- Note that in every block of n transitions starting with a t_{n-i+1}, for $2 \leq i \leq n-1$ we have:

Balanced n-RMGC Construction

contd.

$$\underbrace{t_{n-i_1+1},\overbrace{t_n,\ldots,t_n}^{n-1\text{ times}}}_{\overrightarrow{t_{i_1}}},\ldots,\underbrace{t_{n-i_{(n-1)!}+1},\overbrace{t_n,\ldots,t_n}^{n-1\text{ times}}}_{\overrightarrow{t_{i_{(n-1)!}}}}.$$

Figure: Transition sequence

- From the above two points, the sequence of transitions to be applied to $[n]$ to generate a balanced n-RMGC is as shown in 1
- Note that in every block of n transitions starting with a t_{n-i+1}, for $2 \leq i \leq n-1$ we have:
- the transition t_{n-i+1} with a jump of $n-i+1$

Balanced n-RMGC Construction

contd.

$$\underbrace{t_{n-i_1+1}, \overbrace{t_n, \ldots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_1}}}, \ldots, \underbrace{t_{n-i_{(n-1)!}+1}, \overbrace{t_n, \ldots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_{(n-1)!}}}}.$$

Figure: Transition sequence

- From the above two points, the sequence of transitions to be applied to $[n]$ to generate a balanced n-RMGC is as shown in 1
- Note that in every block of n transitions starting with a t_{n-i+1}, for $2 \leq i \leq n-1$ we have:
- the transition t_{n-i+1} with a jump of $n-i+1$
- $i-1$ transitions of type t_n with a jump of $n+1$

Balanced n-RMGC Construction

contd.

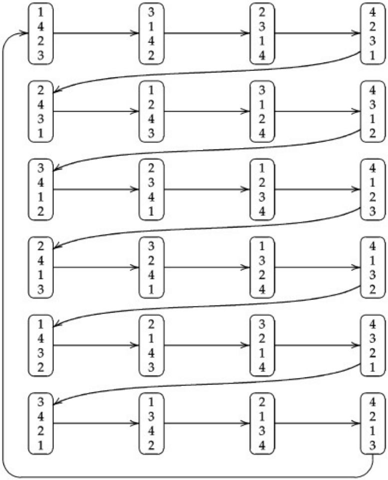
$$\underbrace{t_{n-i_1+1}, \overbrace{t_n, \ldots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_1}}}, \ldots, \underbrace{t_{n-i_{(n-1)!}+1}, \overbrace{t_n, \ldots, t_n}^{n-1 \text{ times}}}_{\overrightarrow{t_{i_{(n-1)!}}}}.$$

Figure: Transition sequence

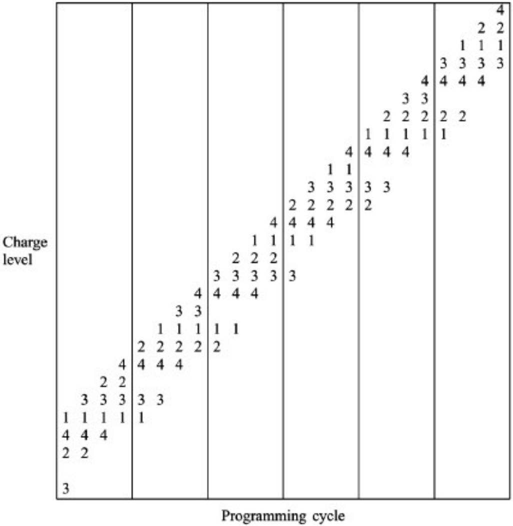
- From the above two points, the sequence of transitions to be applied to $[n]$ to generate a balanced n-RMGC is as shown in 1
- Note that in every block of n transitions starting with a t_{n-i+1}, for $2 \leq i \leq n-1$ we have:
- the transition t_{n-i+1} with a jump of $n-i+1$
- $i-1$ transitions of type t_n with a jump of $n+1$
- $n-i$ transitions of type t_n with a jump of n

Balanced n-RMGC - Example

We now demonstrate the above theorem by ways of an example.



Balanced n-RMGC - Example



Ranking Permutations

For completing the design of a multi-level flash cell, we must define the correspondence between a permutation and its rank in the balanced n-RMGC.

Ranking Permutations

For completing the design of a multi-level flash cell, we must define the correspondence between a permutation and its rank in the balanced n-RMGC. We use the B-Factoradic Number System for representation.

B-Factoradic Number System

A number $m \in \{0, \ldots, n! - 1\}$ can be uniquely represented by the digits $b_{n-1} b_{n-2} \ldots b_1 b_0$ where $b_i \in \{0, \ldots, n - i - 1\}$ and the weight of b_i is $\frac{n!}{(n-i)!}$.

Ranking Permutations - Construction

We now describe an algorithm by way of an example to ascertain the rank of a permutation in the B-Factoradic Number System.

Example 10

- Let $n = 4$ and the current permutation be $\sigma = [1, 4, 3, 2]$, let $pos(k), k \in \{1, \ldots, n\}$ be the position of k in σ from the left. For example, $pos(4) = 2$

Ranking Permutations - Construction

We now describe an algorithm by way of an example to ascertain the rank of a permutation in the B-Factoradic Number System.

Example 10

- Let $n = 4$ and the current permutation be $\sigma = [1, 4, 3, 2]$, let $pos(k), k \in \{1, \ldots, n\}$ be the position of k in σ from the left. For example, $pos(4) = 2$
- We find it's B-Factoradic representation $b_3 b_2 b_1 b_0$ as follows

Ranking Permutations - Construction

We now describe an algorithm by way of an example to ascertain the rank of a permutation in the B-Factoradic Number System.

Example 10

- Let $n = 4$ and the current permutation be $\sigma = [1, 4, 3, 2]$, let $pos(k), k \in \{1, \ldots, n\}$ be the position of k in σ from the left. For example, $pos(4) = 2$
- We find it's B-Factoradic representation $b_3 b_2 b_1 b_0$ as follows
- With $i \in \{0, \ldots n-1\}$, for the i^{th} digit from the right, $b_i =$ the position of the largest element still in the permutation minus 2 (modulo $(n-i)$)

Ranking Permutations - Construction

We now describe an algorithm by way of an example to ascertain the rank of a permutation in the B-Factoradic Number System.

Example 10

- Let $n = 4$ and the current permutation be $\sigma = [1, 4, 3, 2]$, let $pos(k), k \in \{1, \ldots, n\}$ be the position of k in σ from the left. For example, $pos(4) = 2$
- We find it's B-Factoradic representation $b_3 b_2 b_1 b_0$ as follows
- With $i \in \{0, \ldots n-1\}$, for the i^{th} digit from the right, $b_i =$ the position of the largest element still in the permutation minus 2 (modulo $(n-i)$)
- $b_0 = \{pos(4) - 2\}(mod 4) = 0$

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$
- We now recurse on the residual permutation using the steps described above

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$
- We now recurse on the residual permutation using the steps described above
- $b_1 = \{pos(3) - 2\}(mod3) = 1$

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$
- We now recurse on the residual permutation using the steps described above
- $b_1 = \{pos(3) - 2\}(mod 3) = 1$
- $\sigma_2 = \{2, 1\}$ and $b_2 = \{pos(2) - 2\}(mod 2) = 1$

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$
- We now recurse on the residual permutation using the steps described above
- $b_1 = \{pos(3) - 2\}(mod 3) = 1$
- $\sigma_2 = \{2, 1\}$ and $b_2 = \{pos(2) - 2\}(mod 2) = 1$
- $\sigma_3 = \{1\}$ and $b_3 = \{pos(1) - 2\}(mod 1) = 0$

Ranking Permutations - Construction

- Now we remove the largest number from the permutation and read the permutation from the point of removal leftwards, arriving at the residual permutation
- $\sigma = \{1, 4, 3, 2\}, \sigma_1 = \{1, 2, 3\}$
- We now recurse on the residual permutation using the steps described above
- $b_1 = \{pos(3) - 2\}(mod3) = 1$
- $\sigma_2 = \{2, 1\}$ and $b_2 = \{pos(2) - 2\}(mod2) = 1$
- $\sigma_3 = \{1\}$ and $b_3 = \{pos(1) - 2\}(mod1) = 0$
- The B-Factoradic representation is therefore $0_3 1_2 1_1 0_0$

Table of Contents

Conclusion

- We started by looking at what rank modulation is, and why it is needed
- We have looked at a simple construction of cyclic and complete n-RMGC codes
- We then moved to a more efficient balanced n-RMGC construction
- The final mapping from the permutations to the symbols is done by using the B-Factoradic number system

Table of Contents

References

- Anxiao Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," 2008 IEEE International Symposium on Information Theory, 2008, pp. 1731-1735, doi: 10.1109/ISIT.2008.4595284.