Addition of Integers in a Computer Memory : Information Theory Approach

Vladimir B. Balakirsky *

Abstract— The encoding procedure that allows one to represent integers by binary vectors (codewords) in such a way that addition is replaced with the OR operation applied to these vectors is described. The codeword of the sum is constructed using the decoding algorithm. As a result, many of the transformations can be realized using parallel processing, and the method can be considered as a competitor to existing computer arithmetic.

Keywords—encoding, decoding, multi–user communication, addition, parallel processing.

1 Introduction

An important problem, related to the possibility of increasing the size of the working memory of a computer without significant effort, is a realization of some effective trade–off between the size of the memory and the speed of computations. One of the tools for this is a parallelization of the addition of integers. A principle changing of the existing algorithm can be obtained if we define some non–standard representation of the integers by binary vectors (codewords) containing redundancy that allows us to avoid carries which make the operation sequential. We address this problem and, using the properties of the so–called cancellative families of sets [1], describe the procedure where the addition of integers is replaced with the OR operation applied to corresponding codewords constructed as a concatenation of the binary 3–tuples $(0,0,1),(0,1,0),(1,0,0)$ called components of the codeword. Each component encodes the coefficient $-1, 0$, or $+1$ of the representation of the integer by the powers of 3. The binary vector received after the OR operation is transformed to the codeword of the sum (the decoding) using both parallel and sequential operations, where the sequential operation is realized by cyclic shifts of the components of the vector, which is possible because any binary 3–tuple can be obtained from the another binary 3–tuple of the same Hamming weight by a cyclic shift. Hence we cannot completely reach a "component–wise addition", but the length of the vector that should be analyzed sequentially is less than the length of the standard binary representation of integers belonging to the same set by the factor $\log_2 3 = 1.582\ldots$

The difficulties in estimation of the results come with the point that the measure of the efficiency of the procedure is the complexity of transformations in electronics, while the electronics are developed to implement specific methods and play an auxiliary role in this sense. Therefore any conclusions that are based on some theoretical study probably cannot be final. Nevertheless, we believe that if the algorithms allow parallel processing, then this possibility has to be properly understood and used in practice; we also believe that technology can lead to a simpler realization of the cyclic shifts of components of the codewords than adding the carries because of self–synchronizing property of the shifting operation.

* Electrical Engineering Department, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, THE NETHERLANDS. Email : v.b.balakirsky@ele.tue.nl

2 Formal statement of the problem

The standard binary representation of non–negative integers $i \in \{0,\ldots,2^n-1\}$ is known as the function

$$\mathrm{bin}_{n+1}(i) = (b_0,\ldots,b_n) \in \{0,1\}^{n+1}$$

where $b_0,\ldots,b_n \in \{0,1\}$ are uniquely defined by the equation

$$i = \sum_{j=0}^{n} b_j \cdot 2^{n-j}$$

and the standard binary representation of negative integers $i \in \{-2^n,\ldots,-1\}$ is defined as the function

$$\mathrm{bin}_{n+1}(i) = \overline{\mathrm{bin}_{n+1}(-i-1)}$$

where the line above denotes bit–wise inversion. The use of the function bin_{n+1} leads to the following algorithm for adding the integers $m_1, m_2 \in \{-2^n+1,\ldots,2^n-1\}$ having the standard binary representations $(b_0,\ldots,b_n)$ and $(b'_0,\ldots,b'_n)$. The pairs of bits $(c_n,r_n),\ldots,(c_0,r_0)$ are sequentially generated in accordance with the rule

$$(c_j,r_j) = \mathrm{bin}_2(b_j + b'_j + c_{j+1}), \quad j = n,\ldots,0 \tag{1}$$

where $c_{n+1} = 0$ (the bits r_j and c_j are called the result and the carry at the j–th position), and the vector $(r_0,\ldots,r_n)$ determines the standard binary representation of the sum m_1+m_2 in a sense that if $m_1, m_2, m_1+m_2 \in \{-2^n+1,\ldots,2^n-1\}$, then

$$\mathrm{bin}_{n+1}(m_1+m_2) = (r_0,\ldots,r_n).$$

This algorithm can be also viewed as a particular assignment of the functions

$$\begin{aligned}
\mathbf{x}:\ \{-M,\ldots,+M\} &\longrightarrow \{0,1\}^N \\
F:\ \{0,1\}^N \times \{0,1\}^N &\longrightarrow \{0,1\}^N \\
\varphi:\ \{0,1\}^N &\longrightarrow \{-M,\ldots,+M\}
\end{aligned}$$

in the following scheme :

$$\begin{array}{lll} m_1 \longrightarrow \mathbf{x}(m_1) & & \\ \qquad\quad \downarrow & & \\ \qquad F(\mathbf{x}(m_1),\mathbf{x}(m_2)) & \longrightarrow & \varphi(F(\mathbf{x}(m_1),\mathbf{x}(m_2)) \\ \qquad\quad \uparrow & & \\ m_2 \longrightarrow \mathbf{x}(m_2) & & \end{array} \quad (2)$$

Namely, $M = 2^n - 1, N = n+1$, $\mathbf{x}(m) = \mathrm{bin}_{n+1}(m)$, $F(\mathbf{x}(m_1),\mathbf{x}(m_2)) = (r_0,\ldots,r_n)$, and

$$r_0 = 0 \implies \varphi(F(\mathbf{x}(m_1),\mathbf{x}(m_2))) = \sum_{j=1}^{n} r_j \cdot 2^{n-j}$$

$$r_0 = 1 \implies \varphi(F(\mathbf{x}(m_1),\mathbf{x}(m_2))) = -\sum_{j=1}^{n} \overline{r}_j \cdot 2^{n-j} - 1$$

for all $m, m_1, m_2 \in \{-M,\ldots,+M\}$, where $r_0,\ldots,r_n$ are defined in (1) and $\overline{r}_j = 1$ if $r_j = 0$ and $\overline{r}_j = 0$ if $r_j = 1$ for all $j = 1,\ldots,n$.

The transformations given in (2) correspond to data transmission in the scheme where two independent sources communicate with a common receiver over a multiple–access channel. The messages generated by the sources are integers belonging to the set $\{-M,\ldots,+M\}$, they are encoded using the same binary code, and the codewords enter a binary–output multiple–access channel. The receiver has to decode the sum of the integers, i.e., the functions $\mathbf{x}, F$, and φ should be assigned in such a way that if $m_1 + m_2 \in \{-M,\ldots,+M\}$, then

$$\varphi(F(\mathbf{x}(m_1),\mathbf{x}(m_2))) = m_1 + m_2.$$

Since the basic tool of increasing the speed of calculations is parallelization of the algorithm, which can be reached when the operations require the component–wise transformations, we restrict our attention to the functions F defined as

$$F(\mathbf{x},\mathbf{x}') = (f(x_1,x_1'),\ldots,f(x_N,x_N')) \qquad (3)$$

for all $\mathbf{x} = (x_1,\ldots,x_N), \mathbf{x}' = (x_1',\ldots,x_N') \in \{0,1\}^N$, where $f : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ is one of 2^4 Boolean functions specified by the 2×2 binary matrix defining the correspondence between all possible pairs at the input and the bits at the output. Since both rows and columns of the matrix must be distinct (otherwise, the result depends only on one of the terms) and since the matrices obtained one from another by the inversion of all entries are equivalent, we can restrict considerations to two cases : when the matrix is assigned in accordance with the XOR (sum modulo 2) or the OR operation. However, the sum modulo 2 leads to the all–zero vector for every pair of coinciding codewords, and the sum of the corresponding integers is lost. Thus the OR operation,

$$x \vee x' = \begin{cases} 0, & \text{if } (x,x') = (0,0) \\ 1, & \text{if } (x,x') \in \{(0,1),(1,0),(1,1)\} \end{cases}$$

is the only candidate for the function f, and we write $F(\mathbf{x},\mathbf{x}') = \mathbf{x} \vee \mathbf{x}'$. Note also that the requirement (3) can be interpreted in such a way that the multiple–access channel has to be memoryless.

Let us classify any sum $m_1 + m_2 \notin \{-M,\ldots,+M\}$ as the overflow, introduce the function

$$m_1 \boxplus m_2 = \begin{cases} m_1 + m_2 + (2M+1), & \text{if } m_1 + m_2 < -M \\ m_1 + m_2, & \text{if } -M \le m_1 + m_2 \le +M \\ m_1 + m_2 - (2M+1), & \text{if } m_1 + m_2 > +M \end{cases}$$

and set $\varphi(\mathbf{x}(m_1) \vee \mathbf{x}(m_2)) = m_1 \boxplus m_2$. Note also that since the integers exist in the computer memory only via their binary representations, we should modify the scheme (2) in such a way that the sources generate the codewords of the integers and the receiver constructs the codeword of the sum of these integers. More specifically, we will consider the following scheme :

$$\begin{array}{ccc} \mathbf{x}(m_1) & & \\ \downarrow & & \\ \mathbf{x}(m_1) \vee \mathbf{x}(m_2) & \longrightarrow & \mathbf{x}(m_1 \boxplus m_2) \\ \uparrow & & \\ \mathbf{x}(m_2) & & \end{array} \qquad (4)$$

and formulate the problem as *analysis of the scheme (4), which includes the assignment of the encoding function* $\mathbf{x}$ *such that the mapping* $\mathbf{x}(m_1) \vee \mathbf{x}(m_2) \longrightarrow \mathbf{x}(m_1 \boxplus m_2)$ *exists for every pair of integers* $m_1, m_2 \in \{-M,\ldots,+M\}$ *and investigation of its complexity.* Note that the mapping required exists only for very special encoding functions, and that the complexity criterion is rather peculiar in our problem because we cannot measure the effectiveness of "addition" in the number of additions.

3 The encoding algorithm and some properties of the decoding sets

Let us denote

$$\begin{aligned} \mathcal{S}_1 &= \{\, (0,0,1),(0,1,0),(1,0,0) \,\} \\ \mathcal{S}_2 &= \{\, (0,1,1),(1,0,1),(1,1,0) \,\}. \end{aligned}$$

We will also use the left and right cyclic shifts of a given binary vector $y = (y_1,y_2,y_3)$ defined as

$$L[\,y\,] = (y_2,y_3,y_1), \quad R[\,y\,] = (y_3,y_1,y_2) \qquad (5)$$

and note that

$$y,y' \in \mathcal{S}_1 \text{ or } y,y' \in \mathcal{S}_2 \implies y' \in \{y, L[\,y\,], R[\,y\,]\}. \quad (6)$$

Let

$$M = \frac{3^k - 1}{2}$$

where k is some positive integer. Then $2M+1 = 3^k$, and any integer $m \in \{-M,\ldots,+M\}$ can be uniquely represented by the vector

$$\left(m^{(1)},\ldots,m^{(k)}\right) \in \{-1,0,+1\}^k$$

in such a way that

$$m = \sum_{j=1}^{k} m^{(j)} \cdot 3^{k-j}. \quad (7)$$

Let us define *the codeword of the integer* m as

$$\mathbf{x}(m) = \left(x(m^{(1)}), \ldots, x(m^{(k)}) \right) \in \mathcal{S}_1^k \quad (8)$$

where

$$x(m^{(j)}) = \begin{cases} (0,0,1), & \text{if } m^{(j)} = -1 \\ (0,1,0), & \text{if } m^{(j)} = 0 \\ (1,0,0), & \text{if } m^{(j)} = +1 \end{cases} \quad (9)$$

for all $j = 1, \ldots, k$. The 3–tuples $x(m^{(1)}), \ldots, x(m^{(k)})$ will be called *components of the codeword* $\mathbf{x}(m)$. Thus, the result of the OR operation applied to any two codewords is a binary vector of length $3k$ whose components take one of 6 possible values (the 3–tuples $(0,0,0)$ and $(1,1,1)$ never occur). Let

$$\mathcal{D}(m) = \bigcup_{\substack{m_1, m_2 :\, m_1 \boxplus m_2 = m \\ m_1, m_2 \in \{-M, \ldots, +M\}}} \{\, \mathbf{x}(m_1) \vee \mathbf{x}(m_2) \,\} \quad (10)$$

i.e., $\mathbf{y} \in \mathcal{D}(m)$ if and only if there exist $m_1, m_2 \in \{-M, \ldots, +M\}$ such that $m = m_1 \boxplus m_2$ and $\mathbf{y} = \mathbf{x}(m_1) \vee \mathbf{x}(m_2)$. We will call $\mathcal{D}(m)$ *the decoding set of the integer* m.

In the proposition below we claim that if the encoding is defined by (7)–(9), then the OR operation applied to the codewords conserves the sum of the integers in the sense that the pairs of codewords corresponding to the integers having different sums generate distinct vectors. Moreover, the space consisting of 6^k binary vectors of length $3k$ that can be received after the OR operation is partitioned in 3^k groups corresponding to the sums of the arguments and each group contains 2^k vectors. The function $\delta : \mathcal{S}_1 \bigcup \mathcal{S}_2 \to \{-2, \ldots, +2\}$ defined by

$$\begin{pmatrix} y \\ \downarrow \\ \delta(y) \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ -2 & 0 & -1 & +2 & 0 & +1 \end{pmatrix}$$

where the 3–tuples are given in the octal representation, will be used both in the proof of this statement and in the foregoing considerations.

Proposition 1: *If the codeword of any integer* $m \in \{-M, \ldots, +M\}$ *is defined by (7)–(9), then the decoding sets* $\mathcal{D}(-M), \ldots, \mathcal{D}(+M)$ *are pairwise disjoint, i.e., if* $m_1, m_2, m_1', m_2' \in \{-M, \ldots, +M\}$ *are chosen in such a way that* $m_1 \boxplus m_2 \neq m_1' \boxplus m_2'$*, then* $\mathbf{x}(m_1) \vee \mathbf{x}(m_2) \neq \mathbf{x}(m_1') \vee \mathbf{x}(m_2')$. *Furthermore*

$$\bigcup_{m=-M}^{+M} \mathcal{D}(m) = (\mathcal{S}_1 \cup \mathcal{S}_2)^k \quad (11)$$

and

$$|\mathcal{D}(m)| = 2^k, \text{ for all } m \in \{-M, \ldots, +M\}. \quad (12)$$

Proof : If $y^{(j)} = x(m_1^{(j)}) \vee x(m_2^{(j)})$, then

$$\begin{aligned} y^{(j)} \in \mathcal{S}_1 &\implies m_1^{(j)} + m_2^{(j)} = 4 - 2l(y^{(j)}) \\ y^{(j)} \in \mathcal{S}_2 &\implies m_1^{(j)} + m_2^{(j)} = 4 - l_1(y^{(j)}) - l_2(y^{(j)}) \end{aligned}$$

where $l(y^{(j)})$ and $l_1(y^{(j)}), l_2(y^{(j)})$ denote the positions of 1's in $y^{(j)} \in \mathcal{S}_1$ and $y^{(j)} \in \mathcal{S}_2$, respectively. That is the sum $m_1 + m_2$ is uniquely determined by the vector $\mathbf{y} = \mathbf{x}(m_1) \vee \mathbf{x}(m_2)$ in such a way that

$$m_1 + m_2 = \sum_{j=1}^{k} \delta(y^{(j)}) \cdot 3^{k-j}.$$

Hence $m_1 \boxplus m_2$ is also uniquely determined by the vector $\mathbf{y}$, and the first statement follows.

Equation (11) directly follows from (10).

Let us prove (12) using the statement : *for all* $\mathbf{w} = (w^{(1)}, \ldots, w^{(k)}) \in \{1,2\}^k$,

$$\mathbf{y}, \mathbf{y}' \in \mathcal{S}(\mathbf{w}),\ \mathbf{y} \neq \mathbf{y}' \implies \nexists m :\ \mathbf{y}, \mathbf{y}' \in \mathcal{D}(m) \quad (13)$$

where

$$\mathcal{S}(\mathbf{w}) = \Big\{ (y^{(1)}, \ldots, y^{(k)}) \in (\mathcal{S}_1 \cup \mathcal{S}_2)^k : \left(w_H(y^{(1)}), \ldots, w_H(y^{(k)}) \right) = \mathbf{w} \Big\}$$

and w_H *denotes the Hamming weight of the binary vector.*

For any $\mathbf{w} \in \{1,2\}^k$,

$$|\mathcal{S}(\mathbf{w})| = \sum_{m=-M}^{+M} |\mathcal{D}(m) \cap \mathcal{S}(\mathbf{w})| \quad (14)$$

since the decoding sets are pairwise disjoint and their union coincides with $(\mathcal{S}_1 \cup \mathcal{S}_2)^k$. The statement (13) implies $|\mathcal{S}(\mathbf{w}) \cap \mathcal{D}(m)| \leq 1$ for all $m \in \{-M, \ldots, +M\}$, and since $|\mathcal{S}(\mathbf{w})| = 3^k = 2M+1$, equation (14) allows us to replace the inequality with the equality, i.e., we obtain

$$|\mathcal{S}(\mathbf{w}) \cap \mathcal{D}(m)| = 1, \text{ for all } m \in \{-M, \ldots, +M\}.$$

Thus,

$$|\mathcal{D}(m)| = \sum_{\mathbf{w} \in \{1,2\}^k} |\mathcal{D}(m) \cap \mathcal{S}(\mathbf{w})| = 2^k.$$

Proof of (13) : By (10), we conclude that there exists an $m \in \{-M, \ldots, +M\}$ such that $\mathbf{y}, \mathbf{y}' \in \mathcal{D}(m)$ if and only if one can find $m_1, m_2, m_1', m_2' \in \{-M, \ldots, +M\}$ satisfying the equations

$$m_1 \boxplus m_2 = m_1' \boxplus m_2' = m \quad (15)$$

and $\mathbf{y} = \mathbf{x}(m_1) \vee \mathbf{x}(m_2)$, $\mathbf{y}' = \mathbf{x}(m_1') \vee \mathbf{x}(m_2')$. The first equation in (15) implies

$$|m_1 + m_2 - (m_1' + m_2')| \in \{0, 2M+1\}. \quad (16)$$

The integer at the left-hand side of (16) belongs to the set $\{0,\ldots,4M = 2\cdot 3^k - 2\}$, and this integer is uniquely specified by the ternary vector $(\Delta^{(0)},\ldots,\Delta^{(k)}) \in \{0,1,2\}^{k+1}$ whose components are defined by the equation

$$|m_1 + m_2 - (m_1' + m_2')| = \sum_{j=0}^{k} \Delta^{(j)} \cdot 3^{k-j}.$$

Therefore (16) is equivalent to the equalities

$$\Delta^{(1)} = \ldots = \Delta^{(k)} = 0. \tag{17}$$

Let l denote the maximum index j such that $y^{(j)} \neq y^{(j)\prime}$ and let

$$\sigma_l = m_1^{(l)} + m_2^{(l)} - \left(m_1^{(l)\prime} + m_2^{(l)\prime}\right).$$

Then $\sigma_l \in \{\pm 2, \pm 4\}$ if $y^{(l)}, y^{(l)\prime} \in \mathcal{S}_1$ and $\sigma_l = \pm 1$ if $y^{(l)} \in \mathcal{S}_2$ or $y^{(l)\prime} \in \mathcal{S}_2$. In both cases σ_l is not divisible by 3. Since $\Delta^{(l+1)} = \ldots = \Delta^{(k)} = 0$, this observation implies $\Delta^{(l)} \neq 0$, and (17) does not hold. Hence (16) also does not hold, and (13) follows.

4 The decoding algorithm

Since the decoding sets $\mathcal{D}(-M),\ldots,\mathcal{D}(+M)$ are pairwise disjoint, addition of integers can be organized using auxiliary memory where the correspondence between all $6^k = |(\mathcal{S}_1 \cup \mathcal{S}_2)^k|$ possible received vectors and the codewords of the sums is stored. However this procedure can be non–effective because of the memory constraints, and we describe an algorithm that allows us to construct the codeword $\mathbf{x}(m)$ based on any vector $\mathbf{y} \in \mathcal{D}(m) \subset (\mathcal{S}_1 \cup \mathcal{S}_2)^k$ for all $m \in \{-M,\ldots,+M\}$.

The goal of the decoder can be presented as finding a binary vector $\mathbf{z}$ of length $3k$ having the following property :

$$\mathbf{y} \in \mathcal{D}(m) \implies \begin{cases} \mathbf{z} \in \mathcal{S}_1^k \\ \mathbf{z} \vee (0,1,0)^k \in \mathcal{D}(m) \end{cases} \tag{18}$$

where $(0,1,0)^k$ denotes the binary vector of length $3k$ obtained by k repetitions of the 3–tuple $(0,1,0)$; by the definition of the encoding, this vector is the codeword $\mathbf{x}(0)$. Since $\mathbf{y} \in \mathcal{D}(m)$ was generated as $\mathbf{x}(m_1) \vee \mathbf{x}(m_2)$ for some integers $m_1, m_2 \in \{-M,\ldots,+M\}$ chosen in such a way that $m_1 \boxplus m_2 = m$, finding a vector $\mathbf{z}$ satisfying (18) means that the decoder constructs the vector $\mathbf{x}(m) \vee \mathbf{x}(0) \in \mathcal{D}(m)$ and sets $\mathbf{z} = \mathbf{x}(m)$. If $\delta(y^{(1)}),\ldots,\delta(y^{(k)}) \in \{-1,0,+1\}$, then $z^{(1)} = x(\delta(y^{(1)})),\ldots,z^{(k)} = x(\delta(y^{(k)}))$, and the decoding is complete. However $\delta(y^{(j)})$ can be equal to ± 2, while the encoding is defined only for the elements of the set $\{-1,0,+1\}$. Equations (5), (6) can be used to overcome these difficulties and lead to the algorithm where the transformation of the vector $\mathbf{y}$ to the codeword of the sum can be implemented in 3 steps. At the first step, we use a component–wise transformation of $\mathbf{y}$ to the pair of vectors $(\mathbf{z}, \mathbf{c}')$, where $\mathbf{z}$ is the binary vector of length $3k$ consisting of the components belonging to the set $\mathcal{S}_1$, and $\mathbf{c}'$ is the vector of length k containing the elements chosen from the auxiliary alphabet $\{0, L, L', R, R'\}$. Sequential analysis of the vector $\mathbf{c}'$ at the second step leads to the vector $\mathbf{c}^*$ of length $k-1$ that contains the elements belonging to the set $\{0, L, R\}$. Components of the vector $\mathbf{c}'$ specify the modification of the vector $\mathbf{z}$ at the third step, and this transformation is again component–wise. The formal description of the decoding algorithm is presented below, where the binary 3–tuples are given using the octal representation.

(A1) Introduce an alphabet consisting of the symbols $0, L, R, L', R'$ and construct the vectors $\mathbf{z} = \left(z^{(1)},\ldots,z^{(k)}\right) \in \mathcal{S}_1^k$ and $\mathbf{c}' = (c_1',\ldots,c_k') \in \{0, L, R, L', R'\}^k$ using the component–wise mapping

$$\begin{pmatrix} z_j \\ \uparrow \\ y^{(j)} \\ \downarrow \\ c_j' \end{pmatrix} = \begin{pmatrix} 4 & 2 & 1 & 1 & 2 & 4 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 1 & 2 & 3 & 4 & 5 & 6 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ R & 0 & R' & L & 0 & L' \end{pmatrix}$$

for all $j = 1,\ldots,k$.

(A2) Construct the vector $\mathbf{c}^* = (c_2^*,\ldots,c_k^*) \in \{0, L, R\}^{k-1}$ sequentially inspecting the components $c_k',\ldots,c_2'$ of the vector $\mathbf{c}'$ and using the mapping

$$c_j^* = \begin{cases} L, & \text{if } (c_{j+1}^*, c_j') \in \{(0,L),(L,L),(L,L')\} \\ R, & \text{if } (c_{j+1}^*, c_j') \in \{(0,R),(R,R),(R,R')\} \\ 0, & \text{otherwise} \end{cases}$$

for all $j = k,\ldots,2$, where $c_{k+1}^* = 0$.

(A3) Construct the codeword $\left(\tilde{x}^{(1)},\ldots,\tilde{x}^{(k)}\right) \in \mathcal{S}_1^k$ using the component–wise mapping

$$\tilde{x}^{(j)} = \begin{cases} z^{(j)}, & \text{if } c_{j+1}^* = 0 \\ L[\,z^{(j)}\,], & \text{if } c_{j+1}^* = L \\ R[\,z^{(j)}\,], & \text{if } c_{j+1}^* = R \end{cases}$$

for all $j = 1,\ldots,k$.

Proposition 2: *If $\mathbf{y} = \mathbf{x}(m_1) \vee \mathbf{x}(m_2)$ and the codewords $\mathbf{x}(m_1), \mathbf{x}(m_2)$ are defined by (7)–(9), then the (A1)–(A3) decoding algorithm constructs the codeword $\mathbf{x}(m_1 \boxplus m_2)$.*

Proof : Let us describe a recurrent decoding procedure consisting of k steps. At the j–th step, where $j = 1,\ldots,k$, the decoder analyzes $y^{(j)}$ and constructs the vector

$$\mathbf{z}_j = \left(z_j^{(1)},\ldots,z_j^{(k)}\right) \in \mathcal{S}_1^k$$

having the property :

the vector $\mathbf{z}_j \vee (0,1,0)^k$ belongs to the same decoding set as the vector $\left(y^{(1)},\ldots,y^{(j)},(0,1,0)^{k-j}\right)$. (19)

After the k-th step, the decoder has the vector $\mathbf{z} = \mathbf{z}_k$ satisfying (18).

Let us denote $\mathbf{z}_0 = (0,1,0)^k$ and note that this vector satisfies (19) for $j = 0$. Furthermore, for all $j = 1, \ldots, k$, let

$$\mu(\mathbf{z}_{j-1}) = \sum_{i=1}^{k} \mu^{(i)}(\mathbf{z}_{j-1}) \cdot 3^{k-i} \tag{20}$$

where

$$\mu^{(i)}(\mathbf{z}_{j-1}) = \begin{cases} -1, & \text{if } z_{j-1}^{(i)} = (0,0,1) \\ 0, & \text{if } z_{j-1}^{(i)} = (0,1,0) \\ +1, & \text{if } z_{j-1}^{(i)} = (1,0,0) \end{cases} \tag{21}$$

for all $i = 1, \ldots, k$.

Given $j \in \{1, \ldots, k\}$, suppose that $\mathbf{z}_{j-1}$ belongs to the same decoding set as the vector $\left(y^{(1)}, \ldots, y^{(j-1)}, (0,1,0)^{k-j+1}\right)$ and prove that if the vector $\mathbf{z}_j$ is constructed in such a way that

$$\mu(\mathbf{z}_j) = \mu(\mathbf{z}_{j-1}) \boxplus \delta(y^{(j)}) \cdot 3^{k-j} \tag{22}$$

where $\mu(\mathbf{z}_j)$ is defined similar to (20), (21), then (19) holds. Really, m_1, m_2 can be expressed as

$$m_1 = \sum_{j=1}^{k} m_{1j}, \quad m_2 = \sum_{j=1}^{k} m_{2j}$$

where $m_{1j} = m_1^{(j)} \cdot 3^{k-j}$ and $m_{2j} = m_2^{(j)} \cdot 3^{k-j}$ for all $j = 1, \ldots, k$. If $y^{(j)} = x(m_1^{(j)}) \vee x(m_2^{(j)})$, then

$$m_{1j} + m_{2j} = \delta(y^{(j)}) \cdot 3^{k-j} \tag{23}$$

and since $\mathbf{z}_{j-1}$ belongs to the same decoding set as the vector $\left(y^{(1)}, \ldots, y^{(j-1)},\ (0,1,0)^{k-j+1}\right)$ and $x(0) = (0,1,0)$,

$$y^{(i)} = x(\mu^{(i)}(\mathbf{z}_{j-1})) \vee (0,1,0), \quad i = 1, \ldots, j-1.$$

Thus, the integer $\hat{m}$ such that

$$\left(y^{(1)}, \ldots, y^{(j)}, (0,1,0)^{k-j}\right) \in \mathcal{D}(\hat{m})$$

can be generated as

$$\hat{m} = \mu(\mathbf{z}_{j-1}) \boxplus (m_{1j} + m_{2j})$$

and we obtain (22) using (23).

If $y^{(j)} \notin \{(0,0,1),(1,0,0)\}$, then $\delta(y^{(j)}) \in \{-1, 0, +1\}$. Therefore we set

$$z_j^{(i)} = \begin{cases} z_{j-1}^{(i)}, & \text{if } i \neq j \\ x(\delta(y^{(j)})), & \text{if } i = j \end{cases} \tag{24}$$

for all $i = 1, \ldots, k$ and obtain (22), as it follows from the equation $\mu(\mathbf{z}_j) = \mu(\mathbf{z}_{j-1}) + \delta(y^{(j)}) \cdot 3^{k-j}$.

Let us consider the case $y^{(j)} = (0,0,1)$ when $\delta(y^{(j)}) = -2$. For a formal convenience, suppose that $z_{j-1}^{(0)} = (0,1,0)$ and introduce the parameter $t \in \{0, \ldots, j-1\}$ in such a way that $t = j-1$ if $z_{j-1}^{(j-1)} \neq (0,0,1)$ and t is defined by $z_{j-1}^{(t)} \neq (0,0,1)$, $z_{j-1}^{(t+1)} = \ldots = z_{j-1}^{(j-1)} = (0,0,1)$ otherwise. We set

$$z_j^{(i)} = \begin{cases} z_{j-1}^{(i)}, & \text{if } i < t \text{ or } i > j \\ R[\,z_{j-1}^{(i)}\,], & \text{if } i = t \\ (1,0,0), & \text{if } i > t \text{ and } i \leq j \end{cases} \tag{25}$$

for all $i = 1, \ldots, k$, where $R[\,]$ denotes the right cyclic shift defined in (5). Then

$$\mu^{(i)}(\mathbf{z}_j) - \mu^{(i)}(\mathbf{z}_{j-1}) = \begin{cases} 0, & \text{if } i < t \text{ or } i > j \\ -1, & \text{if } i = t \\ +2, & \text{if } i > t \text{ and } i < j \\ +1, & \text{if } i = j \end{cases}$$

where the equality for $i = j$ follows from $z_{j-1}^{(j)} = (0,1,0)$. Therefore if $t \in \{1, \ldots, j-2\}$, then

$$\begin{aligned} &\mu(\mathbf{z}_j) - \mu(\mathbf{z}_{j-1}) \\ &= (-1)3^{k-t} + (+2)\sum_{i=t+1}^{j-1} 3^{k-i} + (+1)3^{k-j} \\ &= (-2) \cdot 3^{k-j}. \end{aligned}$$

If $t = 0$, then

$$\begin{aligned} \mu(\mathbf{z}_j) - \mu(\mathbf{z}_{j-1}) &= (+2)\sum_{i=1}^{j-1} 3^{k-i} + (+1)3^{k-j} \\ &= (-2) \cdot 3^{k-j} + 3^k \end{aligned}$$

and if $t = j-1$, then

$$\begin{aligned} \mu(\mathbf{z}_j) - \mu(\mathbf{z}_{j-1}) &= (-1)3^{k-(j-1)} + (+1)3^{k-j} \\ &= (-2) \cdot 3^{k-j}. \end{aligned}$$

Hence

$$\mu(\mathbf{z}_j) - \mu(\mathbf{z}_{j-1}) \in \left\{ \delta(y^{(j)}) \cdot 3^{k-j},\ 3^k + \delta(y^{(j)}) \cdot 3^{k-j} \right\}$$

and (22) follows.

The analysis of the case $y^{(j)} = (1,0,0)$ is similar : we have to exchange the vectors $(0,0,1)$ and $(1,0,0)$ in the previous considerations and use the left cyclic shift $L[\,]$ instead of $R[\,]$. As a result we obtain

$$\mu(\mathbf{z}_j) - \mu(\mathbf{z}_{j-1}) \in \left\{ \delta(y^{(j)}) \cdot 3^{k-j},\ -3^k + \delta(y^{(j)}) \cdot 3^{k-j} \right\}$$

which also leads to (22).

Summarizing the cases considered above we conclude that if $\mathbf{z} = \mathbf{z}_{j-1}$, then $\mathbf{z}$ can be modified to obtain $\mathbf{z} = \mathbf{z}_j$ using the algorithm below.

- Use the mapping $y^{(j)} \to z^{(j)}$ defined in (A1).
- If $y^{(j)} = (0,0,1)$, then

  (R_1) set $t = j-1$;

  (R_2) if $t = 0$, then go to R_5;

(R_3) cyclically shift the contents of $z^{(t)}$ by 1 position to the right;

(R_4) if $z^{(t)} = (1,0,0)$, then decrease t by 1 and go to R_2;

(R_5) end.

– If $y^{(j)} = (1,0,0)$, then

(L_1) set $t = j-1$;

(L_2) if $t = 0$, then go to L_5;

(L_3) cyclically shift the contents of $z^{(t)}$ by 1 position to the left;

(L_4) if $z^{(t)} = (0,0,1)$, then decrease t by 1 and go to L_2;

(L_5) end.

It is easy to see that the sequential use of this algorithm for all $j = 1,\ldots,k$ leads to the procedure defined by (A1)–(A3), which proves Proposition 2.

5 Examples

Let $k = 4$. Then $M = (3^4-1)/2 = 40$, i.e., we consider the integers belonging to the set $\{-40,\ldots,+40\}$. All binary 3–tuples are given below using the octal representation.

Encoding : $+17 \longrightarrow \mathbf{x}(+17),\ -31 \longrightarrow \mathbf{x}(-31)$.
By (9),

$$\left.\begin{array}{l} +17 = 3^3 - 3^2 - 3^0 \\ -31 = -3^3 - 3^1 - 3^0 \end{array}\right\} \Longrightarrow \left\{\begin{array}{l} \mathbf{x}(+17) = (4,1,2,1) \\ \mathbf{x}(-31) = (1,2,1,1) \end{array}\right.$$

Addition : $\mathbf{x}(+17) \vee \mathbf{x}(-31) \longrightarrow \mathbf{x}(-14)$.
We set

$$\mathbf{y} = \mathbf{x}(+17) \vee \mathbf{x}(-31) = (5,3,3,1)$$

and using the algorithm specified by (A1)–(A3) obtain

(A1) $\mathbf{z} = (2,1,1,4),\ \mathbf{c}' = (0,R',R',R)$

(A2) $\mathbf{c}^* = (R,R,R)$

(A3) $\tilde{\mathbf{x}} = (1,4,4,4)$.

The vector $\tilde{\mathbf{x}}$ is the codeword for the integer

$$(-1)\cdot 3^3 + (+1)\cdot 3^2 + (+1)\cdot 3^1 + (+1)\cdot 3^0 = -14.$$

Let us illustrate the steps of the proof of Proposition 2 for this example. We set $\mathbf{z}_0 = (2,2,2,2)$ and since $y^{(1)}, y^{(2)}, y^{(3)} \notin \{1,4\}$ construct the sequence of vectors $\mathbf{z}_1 = (2,2,2,2)$, $\mathbf{z}_2 = (2,1,2,2)$, $\mathbf{z}_3 = (2,1,1,2)$ using (24) for $j = 1,2,3$. However $y^{(4)} = 1$ and $z_3^{(1)} \neq 1$, $z_3^{(2)} = z_3^{(3)} = 1$ imply that the parameter t, which is needed in (25), is equal to 1. Therefore, by (25), $\mathbf{z}_4 = (R[2],4,4,4) = (1,4,4,4)$ and $\mathbf{z}_4$ is the result of decoding since $k = 4$.

6 Possible extension : multiplication of integers

Similarly to the $\boxplus$ operation, let us introduce the $\boxdot$ operation that replaces the product of integers m_1 and m_2 as follows. Let $q = 0, \pm 1, \pm 2, \ldots$ be defined in such a way that

$$m_1 \cdot m_2 \in \{-M + q\cdot(2M+1), \ldots, +M + q\cdot(2M+1)\}.$$

Then

$$m_1 \boxdot m_2 = m_1 \cdot m_2 - q\cdot(2M+1).$$

Multiplication of integers can be realized as a sequence of additions. If the integers $m_1, m_2 \in \{-M,\ldots,+M\}$ are represented in accordance with (7) by the vectors $(m_1^{(1)},\ldots,m_1^{(k)}), (m_2^{(1)},\ldots,m_2^{(k)}) \in \{-1,0,+1\}^k$, then we may write

$$m_1 \cdot m_2 = \sum_{j=1}^{k} m_2^{(j)} \cdot \left(3^{k-j} \cdot \sum_{i=1}^{k} m_1^{(i)} \cdot 3^{k-i}\right).$$

Since the i–th component of a codeword $\mathbf{x}(m_1)$ encodes $m_1^{(i)}$, $i = 1,\ldots,k$ (see (8), (9)), the multiplication of m_1 by 3^{k-j} corresponds to the $\boxdot$ operation which is realized as the shift of $\mathbf{x}(m_1)$ by $k-j$ components to the left, i.e.,

$$3^{k-j} \boxdot m_1 \quad \longleftrightarrow \quad D^{k-j}[\,\mathbf{x}(m_1)\,]$$

where the l–th component of the vector $D^{k-j}[\,\mathbf{x}(m_1)\,]$ is equal to the $(l+k-j)$–th component of the vector $\mathbf{x}(m_1)$ if $l \le j$ and equal to the binary 3-tuple $(0,1,0)$ if $l > j$; for a formal convenience, we will also use the notation $D^0[\,\mathbf{x}(m_1)\,]$. For a given $j = 1,\ldots,k$, each component of the vector $D^{k-j}[\,\mathbf{x}(m_1)\,]$ is either not changed if $m_2^{(j)} = +1$, or replaced with the vector $(0,1,0)$ if $m_2^{(j)} = 0$, or replaced with its "inverted version" if $m_2^{(j)} = -1$. Thus,

$$\mathbf{x}(m_1 \boxdot m_2) = \\ = \left(m_2^{(1)} * D^{k-1}[\,\mathbf{x}(m_1)\,]\right) \boxplus \ldots \boxplus \left(m_2^{(k)} * D^{0}[\,\mathbf{x}(m_1)\,]\right)$$

where we write $\mathbf{v}' = m * \mathbf{v}$ for all vectors $\mathbf{v}' = (v^{(1)\prime},\ldots,v^{(k)\prime})$, $\mathbf{v} = (v^{(1)},\ldots,v^{(k)}) \in \mathcal{S}_1^k$ constructed in such a way that

$$v^{(j)\prime} = \begin{cases} (0,1,0), & \text{if } m = 0 \text{ or } v^{(j)} = (0,1,0) \\ v^{(j)}, & \text{if } m = +1 \\ (1,0,0), & \text{if } m = -1 \text{ and } v^{(j)} = (0,0,1) \\ (0,0,1), & \text{if } m = -1 \text{ and } v^{(j)} = (1,0,0) \end{cases}$$

for all $j = 1,\ldots,k$. A formal description of the algorithm is presented below.

(M1) Set $j = k-1$ and $\mathbf{z} = m_2^{(k)} * \mathbf{x}(m_1)$.

(M2) If $x(m_2^{(j)}) = (0,1,0)$, then go to (M5).

(M3) Set $\mathbf{y}' = m_2^{(j)} * D^{k-j}[\,\mathbf{x}(m_1)\,]$.

(M4) Set $\mathbf{y} = \mathbf{z} \vee \mathbf{y}'$ and construct the vector $\tilde{\mathbf{x}}$ using the decoding algorithm for the addition. Set $\mathbf{z} = \tilde{\mathbf{x}}$.

(M5) Decrease j by 1. If $j \geq 1$, then go to (M2).

(M6) End : $\mathbf{x}(m_1 \boxdot m_2) = \mathbf{z}$.

References

[1] P. Frankl and Z. Füredi, "Union–free hypergraphs and probability theory," *European Journal of Combinatorics,* vol. 5, pp. 127–131, 1984.