Efficient GNSS secondary code correlations for high sensitivity acquisition

Jérôme Leclère, Maherizo Andrianarison, René Jr Landry
LASSENA, Electrical Engineering Department
École de Technologie Supérieure (ÉTS)
Montréal, Canada
jerome.leclere@lassena.etsmtl.ca

***Abstract*— It is well known that the secondary code introduced in modern GNSS signals complicates the acquisition, because there is a potential sign transition between each period of the primary code. Moreover, if high sensitivity is required, synchronization with the secondary code is necessary to enable long coherent integrations. The simplest solution to synchronize with the secondary code is to combine the primary code correlation results based on the secondary code chips, however this implies a significant computational burden. In a previous article, we proposed several methods to reduce the complexity of the secondary code correlation, namely decomposing the local secondary code in two codes, and using recursion. In this paper, the optimal reduction with these methods is provided, by performing an exhaustive search. It is shown that for the GPS L5 signal, the processing time for a hardware receiver can be reduced by 32 % compared to 20 % previously; the number of operations for the secondary code correlation can be reduced by 69.3 % compared to 65 %, and the processing time with Matlab simulations can be reduced by 45 % compared to 41 %. Improvements are also shown for the Galileo E1 and E5 signals.**

Keywords—GNSS; secondary code; correlation; complexity reduction

I. INTRODUCTION

Most of the modern GNSS signals include a secondary pseudo-random noise code. Although a secondary code brings several advantages and allows a significant improvement of the performance of a GNSS receiver [1], it complicates the acquisition, because it introduces a potential sign transition between each period of the primary code [2,3]. This problem is even more serious when high sensitivity is required, because long coherent integrations are needed, which necessitates the synchronization with the secondary code.

The basic solution to synchronize with the secondary code can be explained as follows : First, the correlation over one period of the primary code is performed (being careful to the potential sign transitions, for example by using zero-padding if fast Fourier transform based circular correlation is used [3]). Then, this step is repeated for several consecutive periods of the primary code. Finally, these correlation results are combined based on the secondary code chips [4]. However, even if the concept of this solution is simple, the computational burden is huge. Especially because the primary codes of the modern GNSS signals are long, have a high chipping rate, or new modulation scheme [5], which implies a significant number of samples to process and requires significant memory to store intermediate results. It is therefore necessary to find methods to reduce the complexity of the secondary code correlation.

In [6], three methods to reduce the complexity have been proposed. The first two methods were in fact based on the same idea, which is to decompose the local secondary code in two codes: one code having a lot of zeros to avoid many operations when computing the correlation; and the second code having a correlation that can be computed efficiently. To allow this efficient computation, the second code should contain a pattern that repeats several times. The third idea was based on recursion, i.e. computing a correlation result using a result previously computed. Note that these methods do not change the operations performed; only the way the operations are performed differs. Therefore, the performance in terms of sensitivity or probabilities of detection and false alarm are strictly the same as with the traditional computation of [4].

In [6], the decompositions for the secondary code were found by inspection. In this paper, an exhaustive search over all the possible decompositions is performed. For the decompositions that give a maximum of zeros in the first code, the minimum complexity of the correlation of the second code is searched. Since these correlations are relatively simple (product of a matrix and a vector with 4 or 5 rows), it is expected that the complexities found are the minimal ones; however, there is no theoretical proof of this.

This paper is organized as follows. Section II describes first the characteristics of the GNSS codes considered, namely the GPS L5 and Galileo E1 OS and E5, because their secondary codes are relatively short. Then the acquisition considered is presented, as well as the secondary code correlation and its complexity. Section III presents the method to reduce the complexity, and explain the methodology for the exhaustive search. The next two sections provide the optimal codes and the corresponding complexity for the GPS L5 and Galileo E1 OS signals, respectively. Section VI discusses the case of the Galileo E5 signal, which is a bit different because there is not only one but 100 secondary codes. Finally, the last section concludes on the performance achievable with this method.

II. GNSS SIGNALS AND THEIR ACQUISITION

A. Pseudo-random noise codes

Modern GNSS signals contain two codes, a primary and a secondary code. The primary code is a binary code (value of +1 or –1) repeating continuously having a high chipping rate, 1.023 Mchip/s or 10.23 Mchip/s most of the time. The secondary code is also a binary code repeating continuously, but it has a much lower chipping rate (1 kchip/s at most). Each chip of the secondary code multiplies one period of the primary code. The product of the primary and secondary codes is then usually called the tiered code [5].

Using matrix notation, denoting $\boldsymbol{c}$ the tiered code, $\boldsymbol{p}$ the primary code and $\boldsymbol{s}$ the secondary code, they can be expressed as

$$\boldsymbol{c} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix}, \quad \boldsymbol{p} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N_\mathrm{P}-1} \end{bmatrix}, \quad \boldsymbol{s} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{N_\mathrm{S}-1} \end{bmatrix}, \tag{1}$$

where N_P is the length of the primary code in sample or chip, N_S the length of the secondary code in chip, and $N = N_\mathrm{P} N_\mathrm{S}$ the length of the tiered code in sample or chip. These codes can be related using the Kronecker product as

$$\boldsymbol{c} = \boldsymbol{s} \otimes \boldsymbol{p} = \begin{bmatrix} s_0 \boldsymbol{p} \\ s_1 \boldsymbol{p} \\ \vdots \\ s_{N_\mathrm{S}-1} \boldsymbol{p} \end{bmatrix}. \tag{2}$$

The characteristics of the primary and secondary codes of the signals considered are given in Tables I and II [7,8]. Note that the secondary code of a GNSS signal is the same for all the satellites, except for the GPS L1C and Galileo E5 signals where each satellite has its own secondary code [8,9].

B. GNSS Acquisition

In this paper, the parallel code search (PCS) acquisition illustrated in Fig. 1 is considered, because it can be easily formulated mathematically. Note however that the methods described in the next sections can be applied to other methods such as parallel frequency search (PFS) [10,11], two-dimensional search [12,13,14], or variation of the PCS [15,16]. Looking at Fig. 1, the incoming signal of length N (length of one period of the tiered code) is multiplied by a local carrier,

TABLE I. CHARACTERISTICS OF PRIMARY CODES ON PILOT CHANNELS

Signal	Chipping rate (Mchip/s)	Length	
		chip	ms
L5	10.230	10 230	1
E1	1.023	4092	4
E5a/b	10.230	10 230	1

TABLE II. CHARACTERISTICS OF SECONDARY CODES ON PILOT CHANNELS. THE VALUE FOR E1 IS PADDED WITH ZEROS AT THE END.

Signal	Chipping rate (chip/s)	Length		Value in hexadecimal
		chip	ms	
L5	1000	20	20	04D4E
E1	250	25	100	380AD90
E5a/b	1000	100	100	Several

and then it is separated into N_S blocks of $2N_\mathrm{P}$ samples (the blocks overlap by N_P samples). These blocks are then zero-padded to reach a length that is a power of two, i.e. $N_\mathrm{FFT} = 2^{\lceil \log_2(2N_\mathrm{P}) \rceil}$. These blocks are denoted $\boldsymbol{x}_i$, with $i = 0, 1, \ldots, N_\mathrm{S} - 1$. Then each block is used to perform a circular correlation with the local zero-padded primary code $\boldsymbol{p}_\mathrm{z}$, giving N_S correlation results of length N_P, denoted $\boldsymbol{r}_i$ [16].

After the correlation with the primary code, assuming the Doppler is completely removed, we obtain

$$\boldsymbol{r}_i = a\, s_{i-m_\mathrm{S}} \boldsymbol{r}_\mathrm{P} + \boldsymbol{\eta}_i, \tag{3}$$

where a is the signal amplitude, s_{i-m_S} is the $(i - m_\mathrm{S})$th secondary code chip (the subscript of s is modulo N_S), m_S is the unknown delay of the incoming secondary code, $\boldsymbol{r}_\mathrm{P}$ is the autocorrelation of the primary code of length N_P, and $\boldsymbol{\eta}_i$ is the noise.

Then the circular correlation with the secondary code is given by

$$\boldsymbol{y}_k = \sum_{i=0}^{N_\mathrm{S}-1} s_{i-k}\, \boldsymbol{r}_i, \tag{4}$$

with $k = 0, 1, \ldots, N_\mathrm{S} - 1$, and where the subscript of s is modulo N_S (the same applies for the next equations when a subscript may be lower than 0 or higher than $N_\mathrm{S} - 1$). Using matrix notation, Eq. (4) can be expressed as

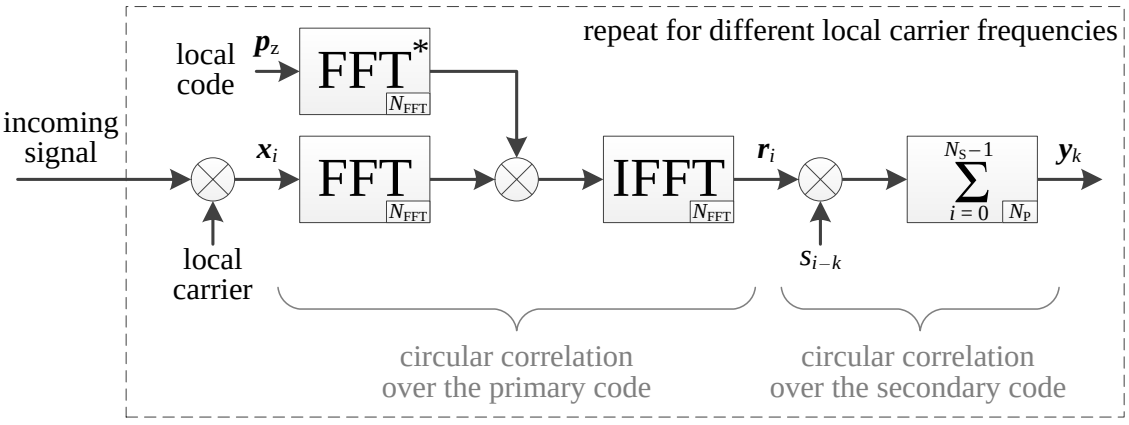

Fig. 1. Parallel code search (PCS) acquisition architecture

$$\begin{bmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_{N_S-1} \end{bmatrix} = \begin{bmatrix} s_0 & s_1 & \cdots & s_{N_S-1} \\ s_{N_S-1} & s_0 & \cdots & s_{N_S-2} \\ \vdots & \vdots & \ddots & \vdots \\ s_1 & s_2 & \cdots & s_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_{N_S-1} \end{bmatrix} \qquad (5)$$
$$\boldsymbol{y} = \boldsymbol{S}\,\boldsymbol{r},$$

where $\boldsymbol{S}$ is a right circulant matrix with $\boldsymbol{s}^{\mathrm{T}}$ as first row. This will be referred to as the traditional implementation.

For hardware implementations, the last block in Fig. 1 is a memory-based accumulator, using one memory of N_{P} elements and one logic adder [16]. Therefore, such accumulator consumes mostly memory.

C. Complexity of secondary code correlation

To evaluate the complexity, the number of operations for software implementations will be estimated. Whereas for hardware implementations, the number of accesses to the input will be estimated, because the processing time is directly proportional to the number of accesses to $\boldsymbol{r}_i$ in hardware [16].

Due to the short length of the secondary codes, computing Eq. (4) or (5) using FFTs would not be efficient [6,16]. Thus, it is better to compute the correlation in a traditional way. Although the inputs are multiplied by the local secondary code, any software or hardware implementation will not implement it as multiplications, but rather as additions or subtractions depending on the value of the chip.

Therefore, computing the output for one delay, i.e. for one row in Eq. (5), requires N_{S} accesses and $N_{\mathrm{S}} - 1$ additions (subtractions are counted as additions). Since there are N_{S} delays, i.e. N_{S} rows in the matrix $\boldsymbol{S}$, there are $N_{\mathrm{S}}N_{\mathrm{S}} = N_{\mathrm{S}}^2$ accesses and $N_{\mathrm{S}}(N_{\mathrm{S}} - 1) = N_{\mathrm{S}}^2 - N_{\mathrm{S}}$ additions in total.

Note that it is possible to reduce the processing time in hardware implementations by simply using two memory-based accumulators to test two delays simultaneously, as shown in Fig. 2. This will be taken into account when comparing the traditional and the proposed implementations.

In order to know the impact of the secondary code correlation on the total processing time, let's estimate the number of operations and cycles required by the implementation given in Fig. 1. For one local carrier frequency tested, the carrier wipe-off requires $2N_{\mathrm{P}}N_{\mathrm{S}}$ multiplications, and the primary code correlation requires $(3N_{\mathrm{FFT}}/2\log_2(N_{\mathrm{FFT}}) + N_{\mathrm{FFT}})N_{\mathrm{S}}$ multiplications and $(3N_{\mathrm{FFT}}\log_2(N_{\mathrm{FFT}}))N_{\mathrm{S}}$ additions. For a hardware implementation, the carrier wipe-off and the primary code correlation can be performed in $N_{\mathrm{FFT}}N_{\mathrm{S}}$ cycles if

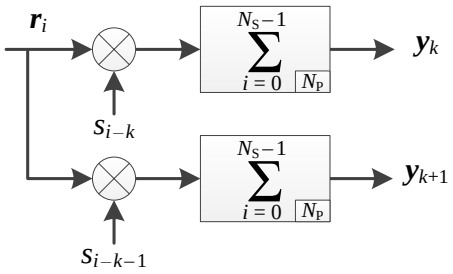

Fig. 2. Traditional implementation using two memory-based accumulators to search simultaneously two delays of the secondary code.

TABLE III. NUMBER OF COMPLEX OPERATIONS AND CYCLES REQUIRED BY THE IMPLEMENTATION OF FIG. 1 FOR THE GPS L5 SIGNAL.

Operation	carrier wipe-off	primary code correlation	secondary code correlation	Total	Portion of SCC
×	819 200	32 768 000	0	33 587 200	0 %
+	0	62 914 560	7 782 400	70 696 960	11.0 %
cycle	1 310 720		8 192 000	9 502 720	86.2 %

a memory is used to store the $\boldsymbol{r}_i$ (omitting the latency) [16]. As example, Table III gives the number of operations and cycles considering the L5 signal with N_{P} = 20 480, N_{FFT} = 65 536 and N_S = 20. It can be seen that the secondary code correlation (SCC) represents only a small portion of the total number of operations. However, it represents the majority of the processing time for a hardware implementation; therefore reducing its processing time will have a major impact. Note nonetheless that when the secondary code becomes longer, the SCC represents a bigger portion (about 40 % of the additions and 30 % of the operations, and more than 96 % of the number of cycles for the E5 signal).

III. PROPOSED METHOD TO REDUCE THE COMPLEXITY

The idea behind the method to reduce the complexity is to express the local secondary code as the sum of two codes, i.e.

$$\boldsymbol{s} = \boldsymbol{s}_{\mathrm{z}} + \boldsymbol{s}_{\Delta}, \qquad (6)$$

with the goal that $\boldsymbol{s}_{\mathrm{z}}$ contains a lot of zeros, and that $\boldsymbol{s}_{\Delta}$ has a correlation simple to compute. For this, $\boldsymbol{s}_{\Delta}$ should contain a pattern, and the length of this pattern should be a divisor of the secondary code length. Indeed, doing so, it is possible to combine several inputs together, and several outputs will be identical. For example, with a code of 6 chips with [1 –1 1] as pattern, the circular correlation is

$$\begin{aligned} \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \\ \boldsymbol{b}_5 \end{bmatrix} &= \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_0 \\ \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \boldsymbol{a}_3 \\ \boldsymbol{a}_4 \\ \boldsymbol{a}_5 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_0 + \boldsymbol{a}_3 \\ \boldsymbol{a}_1 + \boldsymbol{a}_4 \\ \boldsymbol{a}_2 + \boldsymbol{a}_5 \end{bmatrix}, \end{aligned} \qquad (7)$$

where $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are any vectors. Since $\boldsymbol{b}_0 = \boldsymbol{b}_3$, $\boldsymbol{b}_1 = \boldsymbol{b}_4$, $\boldsymbol{b}_2 = \boldsymbol{b}_5$, the operation to compute becomes

$$\begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \\ \boldsymbol{b}_5 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_0 + \boldsymbol{a}_3 \\ \boldsymbol{a}_1 + \boldsymbol{a}_4 \\ \boldsymbol{a}_2 + \boldsymbol{a}_5 \end{bmatrix}. \qquad (8)$$

Thus instead of 6 × 6 = 36 accesses, only 6 + 9 = 15 accesses are needed (6 to combine the inputs, that should be stored in three additional memories, and 9 for the product with the matrix). In return, additional memories are needed to store intermediate results. Therefore, when the proposed implementations will be compared to the traditional one, the number of additional memories needed will be taken into account.

Using (6), the correlation can thus be expressed as

$$\begin{aligned} \boldsymbol{y}_k &= \sum_{i=0}^{N_S-1} \left(s_{z,i-k} + s_{\Delta,i-k}\right) \boldsymbol{r}_i \\ &= \sum_{i=0}^{N_S-1} s_{z,i-k}\, \boldsymbol{r}_i + \sum_{i=0}^{N_S-1} s_{\Delta,i-k}\, \boldsymbol{r}_i \\ &= \boldsymbol{y}_{z,k} + \boldsymbol{y}_{\Delta,k}, \end{aligned} \tag{9}$$

or

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{S}\, \boldsymbol{r} \\ &= (\boldsymbol{S}_z + \boldsymbol{S}_\Delta)\, \boldsymbol{r} \\ &= \boldsymbol{S}_z\, \boldsymbol{r} + \boldsymbol{S}_\Delta\, \boldsymbol{r} \\ &= \boldsymbol{y}_z + \boldsymbol{y}_\Delta, \end{aligned} \tag{10}$$

where $\boldsymbol{S}_z$ and $\boldsymbol{S}_\Delta$ are right-circulant matrices with $\boldsymbol{s}_z^T$ and $\boldsymbol{s}_\Delta^T$ as first row, respectively. As mentioned previously, one or several additional memories will be needed to store $\boldsymbol{y}_\Delta$ completely or partially.

The GPS L5 secondary code has a length of 20 chips, thus $\boldsymbol{s}_\Delta$ should have a pattern length of 1, 2, 4, 5 or 10 (the divisors of 20). The Galileo E1 OS secondary code has a length of 25 chips, thus $\boldsymbol{s}_\Delta$ should have a pattern length of 1 or 5.

Therefore, for these pattern lengths, all the possible codes $\boldsymbol{s}_\Delta$ will be generated, and the number of zeros obtained in the corresponding $\boldsymbol{s}_z$ will be checked. Then, for the most interesting codes $\boldsymbol{s}_\Delta$, the complexity of $\boldsymbol{S}_z\, \boldsymbol{r}$ and $\boldsymbol{S}_\Delta\, \boldsymbol{r}$ will be determined. The codes $\boldsymbol{s}_\Delta$ are considered as binary with +1 and –1 as values, because others values would not make sense since the goal is to obtain zeros in $\boldsymbol{s}_z$.

TABLE IV. STATISTIC OF ALL POSSIBLE s_Δ CODES FOR THE GPS L5 SIGNAL.

Number of zeros in s_z	**Number of patterns for different lengths l**				
	$l = 1$	**$l = 2$**	**$l = 4$**	**$l = 5$**	**$l = 10$**
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	1	0	0
6	0	0	1	2	64
7	0	0	0	0	0
8	1	1	2	7	255
9	0	0	3	0	0
10	0	0	0	12	384
11	0	0	3	0	0
12	1	1	2	7	255
13	0	0	0	0	0
14	0	0	1	2	64
15	0	0	1	0	0

IV. APPLICATION TO THE GPS L5 SIGNAL

A. *Searching of best codes $\boldsymbol{s}_\Delta$*

For a pattern of length 1, there are two possibilities, +1 or –1, i.e. $\boldsymbol{s}_\Delta$ is full of +1 or full of –1. For the other lengths, there are $2^L - 2$ patterns where L is the length (the – 2 is to exclude the case where the code is +1 only or –1 only, which is already the code for a pattern of length 1). Table IV shows how many patterns will provide a certain number of zeros in $\boldsymbol{s}_z$, for the

TABLE V. SELECTED CODES $\boldsymbol{s}_\Delta$ AND $\boldsymbol{s}_z$ FOR THE GPS L5 SIGNAL.

Code ID	Pattern length	Pattern for $\boldsymbol{s}_\Delta$	Codes $\boldsymbol{s}_\Delta$ and $\boldsymbol{s}_z$	Number of zeros in $\boldsymbol{s}_z$
1	1	1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 –2 0 0 –2 –2 0 –2 0 –2 0 0 –2 –2 –2 0	12 (60 %)
2	2	1 –1	1 –1 1 –1 1 –1 1 –1 1 –1 1 –1 1 –1 1 –1 1 –1 1 –1 0 2 0 2 0 0 0 2 –2 0 0 0 0 0 0 2 –2 0 –2 2	12 (60 %)
3	4	1 –1 1 1	1 –1 1 1 1 –1 1 1 1 –1 1 1 1 –1 1 1 1 –1 1 1 0 2 0 0 0 0 0 0 –2 0 0 –2 0 0 0 0 –2 0 –2 0	15 (75 %)
4	4	–1 –1 1 1	–1 –1 1 1 –1 –1 1 1 –1 –1 1 1 –1 –1 1 1 –1 –1 1 1 2 2 0 0 2 0 0 0 0 0 0 –2 2 0 0 0 0 0 –2 0	14 (70 %)
5	5	1 1 1 –1 1	1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 0 0 0 2 0 –2 0 0 0 –2 0 –2 0 0 0 0 –2 –2 0 0	14 (70 %)
6	5	1 –1 1 –1 1	1 –1 1 –1 1 1 –1 1 –1 1 1 –1 1 –1 1 1 –1 1 –1 1 0 2 0 2 0 –2 2 0 0 –2 0 0 0 0 0 0 0 –2 0 0	14 (70 %)
7	10	1 1 1 1 1 –1 –1 –1 –1 –1	1 1 1 1 1 –1 –1 –1 –1 –1 1 1 1 1 1 –1 –1 –1 –1 –1 0 0 0 0 0 0 2 2 0 0 0 –2 0 –2 0 2 0 0 0 2	14 (70 %)

different pattern lengths. The shaded areas show the patterns providing the highest number of zeros for each length. For the patterns with a length of 10, there are 64 such codes, which are shown in Table A.I in the appendix. Among them, the most promising code has been selected. Thus, the list of selected codes $\boldsymbol{s}_\Delta$ and the corresponding $\boldsymbol{s}_z$ is given in Table V. Note that codes 1 and 4 were those provided in [6]. It can thus be seen that the new codes considered have the same number of zeros or more zeros in $\boldsymbol{s}_z$, with a maximum of 15 zeros for the code 3. The next section will evaluate the complexity of their correlation.

B. Correlation complexity for the selected codes

The output is denoted as $\boldsymbol{y} = \boldsymbol{S}_{zj}\,\boldsymbol{r} + \boldsymbol{S}_{\Delta j}\,\boldsymbol{r} = \boldsymbol{y}_{zj} + \boldsymbol{y}_{\Delta j}$, where j is the code ID given in Table V. The use of recursion to compute $\boldsymbol{S}_{zj}\,\boldsymbol{r}$ will be evaluated and applied when it can reduce the complexity (see [6] for details about applying recursion). For this, Table A.II indicates if the recursion is applicable and summarizes the number of accesses required. The number of accesses of all the implementations is summarized in Table A.III.

1) Complexity analysis of code 1 :

The code $\boldsymbol{s}_{z1}$ contains 8 nonzero values (see Table V), thus $8 \times 20 = 160$ accesses and $7 \times 20 = 140$ additions are required to compute the $\boldsymbol{y}_{z1,k}$. The recursion is not interesting because there are 8 or 10 differences between two delayed versions of $\boldsymbol{s}_{z1}$ (see Table A.II).

Regarding $\boldsymbol{y}_{\Delta 1}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 1,0} \\ \boldsymbol{y}_{\Delta 1,1} \\ \vdots \\ \boldsymbol{y}_{\Delta 1,19} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_{19} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_\Sigma \\ \boldsymbol{r}_\Sigma \\ \vdots \\ \boldsymbol{r}_\Sigma \end{bmatrix}, \tag{11}$$

with $\boldsymbol{r}_\Sigma = \sum_{i=0}^{19} \boldsymbol{r}_i$. This requires 20 accesses and 19 additions, and one additional memory to store $\boldsymbol{r}_\Sigma$. Therefore,

$$\boldsymbol{y}_k = \boldsymbol{y}_{z1,k} + \boldsymbol{y}_{\Delta 1,k} = \boldsymbol{y}_{z1,k} + \boldsymbol{r}_\Sigma. \tag{12}$$

The corresponding implementation is shown in Fig. 3, and it requires 180 accesses (160 for the $\boldsymbol{y}_{z1,k}$ and 20 for $\boldsymbol{r}_\Sigma$) and 179 additions (140 for $\boldsymbol{y}_{z1,k}$, 19 for $\boldsymbol{r}_\Sigma$, and 20 to add them) in total.

2) Complexity analysis of code 2 :

The code $\boldsymbol{s}_{z2}$ contains 8 nonzero values and the recursion is not interesting, therefore the complexity to compute $\boldsymbol{y}_{z2}$ is the same as for $\boldsymbol{y}_{z1}$.

Regarding $\boldsymbol{y}_{\Delta 2}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 2,0} \\ \boldsymbol{y}_{\Delta 2,1} \\ \boldsymbol{y}_{\Delta 2,2} \\ \boldsymbol{y}_{\Delta 2,3} \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 & \cdots & -1 \\ -1 & 1 & -1 & 1 & \cdots & 1 \\ 1 & -1 & 1 & -1 & \cdots & -1 \\ -1 & 1 & -1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_{19} \end{bmatrix}, \tag{13}$$

i.e.

$$\begin{aligned} \begin{bmatrix} \boldsymbol{y}_{\Delta 2,2k} \\ \boldsymbol{y}_{\Delta 2,2k+1} \end{bmatrix} &= \begin{bmatrix} 1 & -1 & 1 & -1 & \cdots & -1 \\ -1 & 1 & -1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_{19} \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_2 + \cdots + \boldsymbol{r}_{18} \\ \boldsymbol{r}_1 + \boldsymbol{r}_3 + \cdots + \boldsymbol{r}_{19} \end{bmatrix} \end{aligned} \tag{14}$$

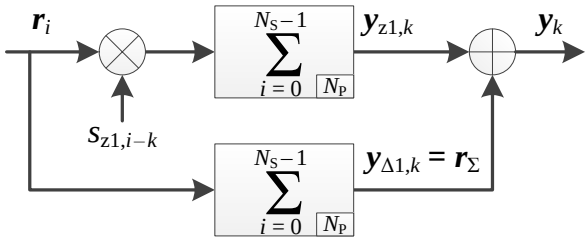

Fig. 3. Implementation of the secondary code correlation using code 1.

$$= \begin{bmatrix} \boldsymbol{r}_{\Sigma\Delta} \\ -\boldsymbol{r}_{\Sigma\Delta} \end{bmatrix},$$

with $\boldsymbol{r}_{\Sigma\Delta} = \sum_{i=0}^{9} \boldsymbol{r}_{2i} - \sum_{i=0}^{9} \boldsymbol{r}_{2i+1}$. This requires 20 accesses and 19 additions. Therefore, the number of accesses and additions is the same for codes 1 and 2, but computing $\boldsymbol{r}_{\Sigma\Delta}$ is slightly more complicated than $\boldsymbol{r}_\Sigma$ because it involves subtractions too, and combining $\boldsymbol{y}_{z2}$ and $\boldsymbol{y}_{\Delta 2}$ is also slightly more complicated than combining $\boldsymbol{y}_{z1}$ and $\boldsymbol{y}_{\Delta 1}$ because it requires additional tests to know if the $\boldsymbol{y}_{\Delta 2}$ part should be added or subtracted. Therefore, this code is not interesting compared to code 1.

3) Complexity analysis of code 3 :

The code $\boldsymbol{s}_{z3}$ contains 6 nonzero values, thus $5 \times 20 = 100$ accesses and $4 \times 20 = 80$ additions are required to compute the $\boldsymbol{y}_{z3,k}$. However, recursion is interesting because $\boldsymbol{y}_{z3,k+10}$ is $\boldsymbol{y}_{z3,k}$ with 4 differences only (see Table A.II). Therefore they can be both computed with $5 + 4 = 9$ accesses, and thus all the $\boldsymbol{y}_{z3,k}$ can be computed with 90 accesses. In software, there are $(4 + 4) \times 10 = 80$ additions (4 to get $\boldsymbol{y}_{z3,k}$ and 4 to get $\boldsymbol{y}_{z3,k+10}$) and 90 memory accesses with recursion. Thus, even if the number of additions is not reduced, since the memory accesses are reduced, the recursion should be beneficial.

Regarding $\boldsymbol{y}_{\Delta 3}$, remembering Eqs. (7) and (8), it is given by

$$\begin{aligned} &\begin{bmatrix} \boldsymbol{y}_{\Delta 3,4k} \\ \boldsymbol{y}_{\Delta 3,4k+1} \\ \boldsymbol{y}_{\Delta 3,4k+2} \\ \boldsymbol{y}_{\Delta 3,4k+3} \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_4 + \boldsymbol{r}_8 + \boldsymbol{r}_{12} + \boldsymbol{r}_{16} \\ \boldsymbol{r}_1 + \boldsymbol{r}_5 + \boldsymbol{r}_9 + \boldsymbol{r}_{13} + \boldsymbol{r}_{17} \\ \boldsymbol{r}_2 + \boldsymbol{r}_6 + \boldsymbol{r}_{10} + \boldsymbol{r}_{14} + \boldsymbol{r}_{18} \\ \boldsymbol{r}_3 + \boldsymbol{r}_7 + \boldsymbol{r}_{11} + \boldsymbol{r}_{15} + \boldsymbol{r}_{19} \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma 3,0} \\ \boldsymbol{r}_{\Sigma 3,1} \\ \boldsymbol{r}_{\Sigma 3,2} \\ \boldsymbol{r}_{\Sigma 3,3} \end{bmatrix} \\ &= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -2 & 0 & 0 \\ 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & -2 \\ -2 & 0 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{r}_{\Sigma 3,0} \\ \boldsymbol{r}_{\Sigma 3,1} \\ \boldsymbol{r}_{\Sigma 3,2} \\ \boldsymbol{r}_{\Sigma 3,3} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{r}_\Sigma \\ \boldsymbol{r}_\Sigma \\ \boldsymbol{r}_\Sigma \\ \boldsymbol{r}_\Sigma \end{bmatrix} - 2 \begin{bmatrix} \boldsymbol{r}_{\Sigma 3,1} \\ \boldsymbol{r}_{\Sigma 3,2} \\ \boldsymbol{r}_{\Sigma 3,3} \\ \boldsymbol{r}_{\Sigma 3,0} \end{bmatrix}, \end{aligned} \tag{15}$$

with $\boldsymbol{r}_{\Sigma 3,k} = \sum_{i=0}^{4} \boldsymbol{r}_{4i+k}$. In software, this requires $4 \times 4 + 3 + 4 = 23$ additions with 5 memories to store $\boldsymbol{r}_\Sigma$ and the $\boldsymbol{r}_{\Sigma 3,k}$. This gives a total of $80 + 23 + 20 = 123$ additions, i.e. a

reduction of 69.25 %. In hardware, the number of accesses required depends on the number of additional memories used.

With three additional memories, 20 accesses are required : 20 accesses to compute and store $\boldsymbol{r}_{\Sigma}$ and $\boldsymbol{r}_{\Sigma 3,1}$ (two memories used). $\boldsymbol{r}_{\Sigma 3,1}$ is used to compute $\boldsymbol{y}_{3,0}$, $\boldsymbol{y}_{3,4}$, $\boldsymbol{y}_{3,8}$, $\boldsymbol{y}_{3,12}$, and $\boldsymbol{y}_{3,16}$. For these computations, all the $\boldsymbol{r}_i$ are accessed, therefore while accessing them $\boldsymbol{r}_{\Sigma 3,2}$ can be computed (third memory used). $\boldsymbol{r}_{\Sigma 3,2}$ is used to compute $\boldsymbol{y}_{3,1}$, $\boldsymbol{y}_{3,5}$, $\boldsymbol{y}_{3,9}$, $\boldsymbol{y}_{3,13}$, and $\boldsymbol{y}_{3,17}$. For these computations, all the $\boldsymbol{r}_i$ are accessed, therefore while accessing them $\boldsymbol{r}_{\Sigma 3,3}$ can be computed that will be saved in the memory containing $\boldsymbol{r}_{\Sigma 3,1}$ (not any more useful). Likewise for $\boldsymbol{r}_{\Sigma 3,0}$.

In software, this cannot be applied because it is assumed that it is not possible to write to different memories simultaneously.

With two additional memories, 35 accesses are required :

- 20 accesses to compute and store $\boldsymbol{r}_{\Sigma}$ and $\boldsymbol{r}_{\Sigma 3,1}$ (two memories used), to compute $\boldsymbol{y}_{3,0}$, $\boldsymbol{y}_{3,4}$, ..., $\boldsymbol{y}_{3,16}$.
- 5 accesses to compute and store $\boldsymbol{r}_{\Sigma 3,2}$ (overwriting $\boldsymbol{r}_{\Sigma 3,1}$), to compute $\boldsymbol{y}_{3,1}$, $\boldsymbol{y}_{3,5}$, ..., $\boldsymbol{y}_{3,17}$.
- 5 accesses to compute and store $\boldsymbol{r}_{\Sigma 3,3}$ (overwriting $\boldsymbol{r}_{\Sigma 3,2}$), to compute $\boldsymbol{y}_{3,2}$, $\boldsymbol{y}_{3,6}$, ..., $\boldsymbol{y}_{3,18}$.
- 5 accesses to compute and store $\boldsymbol{r}_{\Sigma 3,3}$ (overwriting $\boldsymbol{r}_{\Sigma 3,0}$), to compute $\boldsymbol{y}_{3,3}$, $\boldsymbol{y}_{3,7}$, ..., $\boldsymbol{y}_{3,19}$.

In software, this corresponds to 19 + 5 + 5 + 5 + 5 = 39 additions with two memories.

With only one additional memory, 50 accesses are required :

- 20 accesses to compute and store $\boldsymbol{r}_{\Sigma} - 2\boldsymbol{r}_{\Sigma 3,1}$, to compute $\boldsymbol{y}_{3,0}$, $\boldsymbol{y}_{3,4}$, ..., $\boldsymbol{y}_{3,16}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 3,1}$ and subtract 2 $\boldsymbol{r}_{\Sigma 3,2}$, to compute $\boldsymbol{y}_{3,1}$, $\boldsymbol{y}_{3,5}$, ..., $\boldsymbol{y}_{3,17}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 3,2}$ and subtract 2 $\boldsymbol{r}_{\Sigma 3,3}$, to compute $\boldsymbol{y}_{3,2}$, $\boldsymbol{y}_{3,6}$, ..., $\boldsymbol{y}_{3,18}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 3,3}$ and subtract 2 $\boldsymbol{r}_{\Sigma 3,0}$, to compute $\boldsymbol{y}_{3,3}$, $\boldsymbol{y}_{3,7}$, ..., $\boldsymbol{y}_{3,19}$.

In software, this corresponds to 19 + 10 + 10 + 10 = 49 additions with one memory.

In conclusion, with one additional memory, there is a total of 90 + 50 = 140 accesses, which is lower than the 180 accesses of the code 1. Code 3 is thus better than code 1.

4) Complexity analysis of code 4 :

The code $\boldsymbol{s}_{z4}$ contains 6 nonzero values, thus 6 × 20 = 120 accesses and 5 × 20 = 100 additions are required to compute the $\boldsymbol{y}_{z4,k}$. The recursion is not interesting because there are 8 or 9 differences between two delayed versions of $\boldsymbol{s}_{z4}$.

Regarding $\boldsymbol{y}_{\Delta 4}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 4,4k} \\ \boldsymbol{y}_{\Delta 4,4k+1} \\ \boldsymbol{y}_{\Delta 4,4k+2} \\ \boldsymbol{y}_{\Delta 4,4k+3} \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_4 + \boldsymbol{r}_8 + \boldsymbol{r}_{12} + \boldsymbol{r}_{16} \\ \boldsymbol{r}_1 + \boldsymbol{r}_5 + \boldsymbol{r}_9 + \boldsymbol{r}_{13} + \boldsymbol{r}_{17} \\ \boldsymbol{r}_2 + \boldsymbol{r}_6 + \boldsymbol{r}_{10} + \boldsymbol{r}_{14} + \boldsymbol{r}_{18} \\ \boldsymbol{r}_3 + \boldsymbol{r}_7 + \boldsymbol{r}_{11} + \boldsymbol{r}_{15} + \boldsymbol{r}_{19} \end{bmatrix} \quad (16)$$

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 4,4k} \\ \boldsymbol{y}_{\Delta 4,4k+1} \\ \boldsymbol{y}_{\Delta 4,4k+2} \\ \boldsymbol{y}_{\Delta 4,4k+3} \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_4 + \boldsymbol{r}_8 + \boldsymbol{r}_{12} + \boldsymbol{r}_{16} \ldots \\ -\boldsymbol{r}_2 - \boldsymbol{r}_6 - \boldsymbol{r}_{10} - \boldsymbol{r}_{14} - \boldsymbol{r}_{18} \\ \boldsymbol{r}_1 + \boldsymbol{r}_5 + \boldsymbol{r}_9 + \boldsymbol{r}_{13} + \boldsymbol{r}_{17} \ldots \\ -\boldsymbol{r}_3 - \boldsymbol{r}_7 - \boldsymbol{r}_{11} - \boldsymbol{r}_{15} - \boldsymbol{r}_{19} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma\Delta 4,0} \\ \boldsymbol{r}_{\Sigma\Delta 4,1} \end{bmatrix},$$

with $\boldsymbol{r}_{\Sigma\Delta 4,0} = \sum_{i=0}^{4} -\boldsymbol{r}_{4i} - \boldsymbol{r}_{4i+1} + \boldsymbol{r}_{4i+2} + \boldsymbol{r}_{4i+3}$, and $\boldsymbol{r}_{\Sigma\Delta 4,1} = \sum_{i=0}^{4} \boldsymbol{r}_{4i} - \boldsymbol{r}_{4i+1} - \boldsymbol{r}_{4i+2} + \boldsymbol{r}_{4i+3}$. This requires 20 accesses with two memories (to store $\boldsymbol{r}_{\Sigma\Delta 4,0}$ and $\boldsymbol{r}_{\Sigma\Delta 4,1}$), or 40 accesses with one memory (to store $\boldsymbol{r}_{\Sigma\Delta 4,0}$ to compute $\boldsymbol{y}_{\Delta 4,4k}$ and $\boldsymbol{y}_{\Delta 4,4k+2}$, and then to store $\boldsymbol{r}_{\Sigma\Delta 4,1}$ to compute $\boldsymbol{y}_{\Delta 4,4k+1}$ and $\boldsymbol{y}_{\Delta 4,4k+3}$). In software, this corresponds to 20 additions, giving a total of 140 additions (reduction of 65 %), which is higher than with code 3.

However, Eq. (16) can also be written as

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 4,4k} \\ \boldsymbol{y}_{\Delta 4,4k+1} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{y}_{\Delta 4,4k+2} \\ \boldsymbol{y}_{\Delta 4,4k+3} \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma 3,0} - \boldsymbol{r}_{\Sigma 3,2} \\ \boldsymbol{r}_{\Sigma 3,1} - \boldsymbol{r}_{\Sigma 3,3} \end{bmatrix}. \quad (17)$$

With only one additional memory, 30 accesses are required to compute Eq. (17) :

- 20 accesses to compute and store $-\left(\boldsymbol{r}_{\Sigma 3,0} - \boldsymbol{r}_{\Sigma 3,2}\right) - \left(\boldsymbol{r}_{\Sigma 3,1} - \boldsymbol{r}_{\Sigma 3,3}\right)$, to compute $\boldsymbol{y}_{4,0}$, $\boldsymbol{y}_{4,2}$, ..., $\boldsymbol{y}_{4,18}$.
- 10 accesses to add 2 $\left(\boldsymbol{r}_{\Sigma 3,1} - \boldsymbol{r}_{\Sigma 3,3}\right)$, to compute $\boldsymbol{y}_{4,1}$, $\boldsymbol{y}_{4,3}$, ..., $\boldsymbol{y}_{3,19}$.

This is 10 accesses fewer than with Eq. (16). Note that this last formulation was not found in [6].

In conclusion, with one additional memory, there is a total of 120 + 30 = 150 accesses, which is a bit higher than the 140 accesses of the code 3. Code 4 is thus less good than code 3.

5) Complexity analysis of code 5 :

The code $\boldsymbol{s}_{z5}$ contains also 6 nonzero values, thus 120 accesses and 100 additions are required to compute $\boldsymbol{y}_{z5}$. The recursion is not interesting because there are 8 or 9 differences between two delayed versions of $\boldsymbol{s}_{z5}$.

Regarding $\boldsymbol{y}_{\Delta 5}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 5,5k} \\ \boldsymbol{y}_{\Delta 5,5k+1} \\ \boldsymbol{y}_{\Delta 5,5k+2} \\ \boldsymbol{y}_{\Delta 5,5k+3} \\ \boldsymbol{y}_{\Delta 5,5k+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_5 + \boldsymbol{r}_{10} + \boldsymbol{r}_{15} \\ \boldsymbol{r}_1 + \boldsymbol{r}_6 + \boldsymbol{r}_{11} + \boldsymbol{r}_{16} \\ \boldsymbol{r}_2 + \boldsymbol{r}_7 + \boldsymbol{r}_{12} + \boldsymbol{r}_{17} \\ \boldsymbol{r}_3 + \boldsymbol{r}_8 + \boldsymbol{r}_{13} + \boldsymbol{r}_{18} \\ \boldsymbol{r}_4 + \boldsymbol{r}_9 + \boldsymbol{r}_{14} + \boldsymbol{r}_{19} \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{r}_{\Sigma} \\ \boldsymbol{r}_{\Sigma} \\ \boldsymbol{r}_{\Sigma} \\ \boldsymbol{r}_{\Sigma} \\ \boldsymbol{r}_{\Sigma} \end{bmatrix} - 2\begin{bmatrix} \boldsymbol{r}_{\Sigma 5,3} \\ \boldsymbol{r}_{\Sigma 5,4} \\ \boldsymbol{r}_{\Sigma 5,0} \\ \boldsymbol{r}_{\Sigma 5,1} \\ \boldsymbol{r}_{\Sigma 5,2} \end{bmatrix}, \quad (18)$$

with $\boldsymbol{r}_{\Sigma 5,k} = \sum_{i=0}^{4} \boldsymbol{r}_{5i+k}$. This operation looks like the one with code 3. With one additional memory, 52 accesses are required:

- 20 accesses to compute and store $\boldsymbol{r}_{\Sigma} - 2\boldsymbol{r}_{\Sigma 5,3}$, to compute $\boldsymbol{y}_{\Delta 5,5k}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 5,3}$ and subtract 2 $\boldsymbol{r}_{\Sigma 5,4}$, to compute $\boldsymbol{y}_{\Delta 5,5k+1}$.

- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 5,4}$ and subtract 2 $\boldsymbol{r}_{\Sigma 5,0}$, to compute $\boldsymbol{y}_{\Delta 5,5k+2}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 5,0}$ and subtract 2 $\boldsymbol{r}_{\Sigma 5,1}$, to compute $\boldsymbol{y}_{\Delta 5,5k+3}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 5,1}$ and subtract 2 $\boldsymbol{r}_{\Sigma 5,2}$, to compute $\boldsymbol{y}_{\Delta 5,5k+4}$.

In conclusion, with one additional memory, there is a total of 120 + 52 = 172 accesses, which is higher than the 140 accesses of the code 3. Code 5 is thus less good than code 3.

6) Complexity analysis of code 6 :

The code $\boldsymbol{s}_{z6}$ contains also 6 nonzero values, thus 120 accesses and 100 additions are required to compute $\boldsymbol{y}_{z6}$. The recursion is not interesting because there are 8 or 9 differences between two delayed versions of $\boldsymbol{s}_{z6}$.

Regarding $\boldsymbol{y}_{\Delta 6}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 6,5k} \\ \boldsymbol{y}_{\Delta 6,5k+1} \\ \boldsymbol{y}_{\Delta 6,5k+2} \\ \boldsymbol{y}_{\Delta 6,5k+3} \\ \boldsymbol{y}_{\Delta 6,5k+4} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_5 + \boldsymbol{r}_{10} + \boldsymbol{r}_{15} \\ \boldsymbol{r}_1 + \boldsymbol{r}_6 + \boldsymbol{r}_{11} + \boldsymbol{r}_{16} \\ \boldsymbol{r}_2 + \boldsymbol{r}_7 + \boldsymbol{r}_{12} + \boldsymbol{r}_{17} \\ \boldsymbol{r}_3 + \boldsymbol{r}_8 + \boldsymbol{r}_{13} + \boldsymbol{r}_{18} \\ \boldsymbol{r}_4 + \boldsymbol{r}_9 + \boldsymbol{r}_{14} + \boldsymbol{r}_{19} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma 6,0} \\ \boldsymbol{r}_{\Sigma 6,1} \\ \boldsymbol{r}_{\Sigma 6,2} \\ \boldsymbol{r}_{\Sigma 6,3} \\ \boldsymbol{r}_{\Sigma 6,4} \end{bmatrix}, \quad (19)$$

with $\boldsymbol{r}_{\Sigma 6,k} = \sum_{i=0}^{3} \boldsymbol{r}_{5i+k}$. Recursion can be used, since two rows differ by two elements if computed in one of these orders : $\boldsymbol{y}_{\Delta 6,0}, \boldsymbol{y}_{\Delta 6,2}, \boldsymbol{y}_{\Delta 6,4}, \boldsymbol{y}_{\Delta 6,1}, \boldsymbol{y}_{\Delta 6,3}$ or $\boldsymbol{y}_{\Delta 6,0}, \boldsymbol{y}_{\Delta 6,3}, \boldsymbol{y}_{\Delta 6,1}, \boldsymbol{y}_{\Delta 6,4}, \boldsymbol{y}_{\Delta 6,2}$.

With one additional memory, 52 accesses are required :

- 20 accesses to compute and store $\boldsymbol{y}_{\Delta 6,5k}$, to compute $\boldsymbol{y}_{6,0}$, $\boldsymbol{y}_{6,5}$, $\boldsymbol{y}_{6,10}$, $\boldsymbol{y}_{6,15}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 6,1}$ and subtract 2 $\boldsymbol{r}_{\Sigma 6,0}$ to get $\boldsymbol{y}_{\Delta 6,2}$ to compute $\boldsymbol{y}_{6,2}$, $\boldsymbol{y}_{6,7}$, $\boldsymbol{y}_{6,12}$, $\boldsymbol{y}_{6,17}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 6,3}$ and subtract 2 $\boldsymbol{r}_{\Sigma 6,2}$ to get $\boldsymbol{y}_{\Delta 6,4}$ to compute $\boldsymbol{y}_{6,4}$, $\boldsymbol{y}_{6,9}$, $\boldsymbol{y}_{6,14}$, $\boldsymbol{y}_{6,19}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 6,0}$ and subtract 2 $\boldsymbol{r}_{\Sigma 6,4}$ to get $\boldsymbol{y}_{\Delta 6,1}$ to compute $\boldsymbol{y}_{6,1}$, $\boldsymbol{y}_{6,6}$, $\boldsymbol{y}_{6,11}$, $\boldsymbol{y}_{6,16}$.
- 8 accesses to add 2 $\boldsymbol{r}_{\Sigma 6,2}$ and subtract 2 $\boldsymbol{r}_{\Sigma 6,1}$ to get $\boldsymbol{y}_{\Delta 6,3}$ to compute $\boldsymbol{y}_{6,3}$, $\boldsymbol{y}_{6,8}$, $\boldsymbol{y}_{6,13}$, $\boldsymbol{y}_{6,18}$.

In conclusion, the code 6 has the same complexity as the code 5, and it is thus less good than the code 3.

7) Complexity analysis of code 7 :

The code $\boldsymbol{s}_{z7}$ contains also 6 nonzero values, thus 120 accesses and 100 additions are required to compute $\boldsymbol{y}_{z7}$. The recursion is not interesting because there are 7 or 9 differences between two delayed versions of $\boldsymbol{s}_{z7}$.

Regarding $\boldsymbol{y}_{\Delta 7}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 7,10k} \\ \boldsymbol{y}_{\Delta 7,10k+1} \\ \boldsymbol{y}_{\Delta 7,10k+2} \\ \boldsymbol{y}_{\Delta 7,10k+3} \\ \boldsymbol{y}_{\Delta 7,10k+4} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{y}_{\Delta 7,10k+5} \\ \boldsymbol{y}_{\Delta 7,10k+6} \\ \boldsymbol{y}_{\Delta 7,10k+7} \\ \boldsymbol{y}_{\Delta 7,10k+8} \\ \boldsymbol{y}_{\Delta 7,10k+9} \end{bmatrix} \quad (20)$$

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 7,10k} \\ \boldsymbol{y}_{\Delta 7,10k+1} \\ \boldsymbol{y}_{\Delta 7,10k+2} \\ \boldsymbol{y}_{\Delta 7,10k+3} \\ \boldsymbol{y}_{\Delta 7,10k+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma\Delta 7,0} \\ \boldsymbol{r}_{\Sigma\Delta 7,1} \\ \boldsymbol{r}_{\Sigma\Delta 7,2} \\ \boldsymbol{r}_{\Sigma\Delta 7,3} \\ \boldsymbol{r}_{\Sigma\Delta 7,4} \end{bmatrix}$$

with $\boldsymbol{r}_{\Sigma\Delta 7,k} = \boldsymbol{r}_k - \boldsymbol{r}_{5+k} + \boldsymbol{r}_{10+k} - \boldsymbol{r}_{15+k}$. Using recursion, this requires 36 accesses with one memory :

- 20 accesses to compute and store the sum of the $\boldsymbol{r}_{\Sigma\Delta 7,k}$, to compute $\boldsymbol{y}_{\Delta 7,10k}$ and $\boldsymbol{y}_{\Delta 7,10k+5}$.
- 4 accesses to subtract 2 $\boldsymbol{r}_{\Sigma\Delta 7,0}$, to compute $\boldsymbol{y}_{\Delta 7,10k+1}$ and $\boldsymbol{y}_{\Delta 7,10k+6}$.
- 4 accesses to subtract 2 $\boldsymbol{r}_{\Sigma 7,1}$, to compute $\boldsymbol{y}_{\Delta 7,10k+2}$ and $\boldsymbol{y}_{\Delta 7,10k+7}$.
- 4 accesses to subtract 2 $\boldsymbol{r}_{\Sigma 7,2}$, to compute $\boldsymbol{y}_{\Delta 7,10k+3}$ and $\boldsymbol{y}_{\Delta 7,10k+8}$.
- 4 accesses to subtract 2 $\boldsymbol{r}_{\Sigma 7,3}$, to compute $\boldsymbol{y}_{\Delta 7,10k+4}$ and $\boldsymbol{y}_{\Delta 7,10k+9}$.

In software, this corresponds to 19 + 4 × 4 = 35 additions with one memory, or to 15 + 4 + 4 = 23 additions with 5 memories.

In conclusion, with one additional memory, there are a total of 120 + 36 = 156 accesses, which is higher than the 140 accesses of the code 3. Code 7 is thus less good than code 3.

C. Complexity analysis summary

Table A.III summarizes the number of accesses for all the codes and provides the complexity reduction compared to the traditional implementation. The column reduction indicates two percentages corresponding to the ratio of number of accesses compared to the traditional implementation: the first one is the traditional implementation without additional memory, and the second one the traditional implementation with the same number of additional memories. Therefore, the second percentage gives the fair comparison for hardware implementations where memories are important. As for the first one, it makes sense for software implementations, because the number of additions is closely related to the number of accesses with the proposed methods, whereas the traditional implementation always have the same number of additions whatever the number of additional memories used.

The eighth column is the number of accesses using an additional trick. For example, considering the code 1 (Eq. (12) and Fig. 3), when accessing the first 20 $\boldsymbol{r}_i$ to compute $\boldsymbol{r}_\Sigma$, it is possible to compute $\boldsymbol{y}_{z1,0}$ at the same time, and thus to save 8 accesses for a total of 172 accesses. It requires only some additional control logic to do it. Note that this trick cannot be applied with the traditional implementation, nor to software implementation.

As a reminder, codes 1 and 4 were already proposed in [5], and Table A.III shows that these were indeed better than the traditional method. After the exhaustive search, the complexity for the code 4 has been reduced, and one better code has been found, code 3. The reduction can be up to 69.3 %, and if only one additional memory is used, the hardware implementation can be 32 % faster compared to 20 % previously with the code 4 (without the trick mentioned above, and with the non-optimal reduction proposed in [6]). Note however, that for hardware

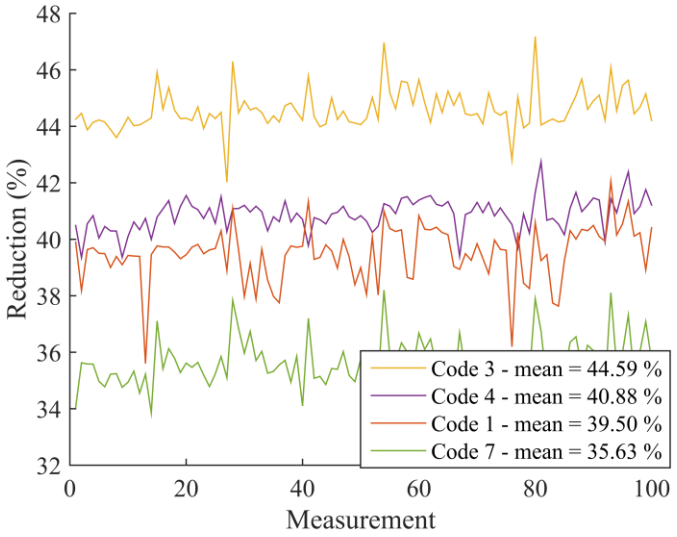

Fig. 4. Reduction of the processing time under Matlab for the different codes, for the GPS L5 signal.

implementation with at least one additional memory, the best code is still less efficient than the recursion applied directly to the L5 secondary code (Method #3 of [6]). Indeed, with one additional memory, the recursion requires only 94 accesses, i.e. a reduction of 53 % compared to the traditional implementation with one additional memory.

Table A.III also shows that when more than one additional memory is used, the improvement is much less, or even not possible such as with three additional memories.

In order to have a more accurate estimation of the performance in software, the reduction of the processing time has been estimated with Matlab simulations. The processing time has been measured over 200 iterations of the algorithms, and 100 measurements have been performed. Fig. 4 shows the results. Codes 1 and 4 offer a reduction of the processing time of about 40 % and 41 %, as already found in [6]. Code 3 offers a higher reduction with about 45 % (without using recursion, else the reduction is a bit lower), as expected, while the code 7 offers a lower reduction, even if it is supposed to be better than code 1. This is probably due the fact that more conditions are performed, while code 1 is more straightforward to implement, even if it requires more additions or subtraction.

V. Application to the Galileo E1 OS signal

A. *Searching of best codes* $\boldsymbol{s}_\Delta$

The E1 secondary codes having 15 ones, the pattern +1 of length 1 provides 15 zeros in $\boldsymbol{s}_z$. For the pattern of length 5, 4 codes provide a higher number of zeros in $\boldsymbol{s}_z$. These codes are summarized in Table VI. Note that the code 1 was already provided in [6]. The recursion is not interesting for any of the codes $\boldsymbol{s}_z$, and thus not mention later.

B. *Correlation complexity for the selected codes*

1) Complexity analysis of code 1 :

The code 1 is the same as the code 1 of GPS L5, therefore Eq. (12) applies here also, giving a total of 10 × 25 + 25 = 275 accesses and 274 additions.

2) Complexity analysis of code 2 :

The code $\boldsymbol{s}_{z2}$ contains 8 nonzero values (see Table VI), thus 8 × 25 = 200 accesses and 7 × 25 = 175 additions are required to compute the $\boldsymbol{y}_{z2,k}$.

Regarding $\boldsymbol{y}_{\Delta 2}$, it is given by

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 2,5k} \\ \boldsymbol{y}_{\Delta 2,5k+1} \\ \boldsymbol{y}_{\Delta 2,5k+2} \\ \boldsymbol{y}_{\Delta 2,5k+3} \\ \boldsymbol{y}_{\Delta 2,5k+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 + \boldsymbol{r}_5 + \boldsymbol{r}_{10} + \boldsymbol{r}_{15} + \boldsymbol{r}_{20} \\ \boldsymbol{r}_1 + \boldsymbol{r}_6 + \boldsymbol{r}_{11} + \boldsymbol{r}_{16} + \boldsymbol{r}_{21} \\ \boldsymbol{r}_2 + \boldsymbol{r}_7 + \boldsymbol{r}_{12} + \boldsymbol{r}_{17} + \boldsymbol{r}_{22} \\ \boldsymbol{r}_3 + \boldsymbol{r}_8 + \boldsymbol{r}_{13} + \boldsymbol{r}_{18} + \boldsymbol{r}_{23} \\ \boldsymbol{r}_4 + \boldsymbol{r}_9 + \boldsymbol{r}_{14} + \boldsymbol{r}_{19} + \boldsymbol{r}_{24} \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_{\Sigma 2,0} \\ \boldsymbol{r}_{\Sigma 2,1} \\ \boldsymbol{r}_{\Sigma 2,2} \\ \boldsymbol{r}_{\Sigma 2,3} \\ \boldsymbol{r}_{\Sigma 2,4} \end{bmatrix}, \quad (21)$$

with $\boldsymbol{r}_{\Sigma 2,k} = \sum_{i=0}^{4} \boldsymbol{r}_{5i+k}$. Recursion can be used, since they differ by two elements if computed in one of these orders : $\boldsymbol{y}_{\Delta 2,0}, \boldsymbol{y}_{\Delta 2,2}, \boldsymbol{y}_{\Delta 2,4}, \boldsymbol{y}_{\Delta 2,1}, \boldsymbol{y}_{\Delta 2,3}$ or $\boldsymbol{y}_{\Delta 2,0}, \boldsymbol{y}_{\Delta 2,3}, \boldsymbol{y}_{\Delta 2,1}, \boldsymbol{y}_{\Delta 2,4}, \boldsymbol{y}_{\Delta 2,2}$.

With one additional memory, 65 accesses are required :

- 25 accesses to compute and store $\boldsymbol{y}_{\Delta 2,0}$, to compute $\boldsymbol{y}_{2,0}$, $\boldsymbol{y}_{2,5}$, $\boldsymbol{y}_{2,10}$, $\boldsymbol{y}_{2,15}$.

TABLE VI. Selected codes $\boldsymbol{s}_\Delta$ and $\boldsymbol{s}_z$ for the Galileo E1 OS signal.

Code ID	Pattern length	Pattern for $\boldsymbol{s}_\Delta$	Codes $\boldsymbol{s}_\Delta$ and $\boldsymbol{s}_z$	Number of zeros in $\boldsymbol{s}_z$
1	1	1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	15 (60 %)
			0 0 –2 –2 –2 0 0 0 0 0 0 0 –2 0 –2 0 –2 –2 0 –2 –2 0 0 –2 0	
2	5	1 1 –1 1 –1	1 1 –1 1 –1 1 1 –1 1 –1 1 1 –1 1 –1 1 1 –1 1 –1 1 1 –1 1 –1	17 (68 %)
			0 0 0 –2 0 0 0 2 0 2 0 0 0 0 0 0 –2 0 0 0 –2 0 2 –2 2	
3	5	1 1 1 1 –1	1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1	16 (64 %)
			0 0 –2 –2 0 0 0 0 0 2 0 0 –2 0 0 0 –2 –2 0 0 –2 0 0 –2 2	
4	5	1 1 –1 1 1	1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1 1 1 –1 1 1	16 (64 %)
			0 0 0 –2 –2 0 0 2 0 0 0 0 0 0 –2 0 –2 0 0 –2 –2 0 2 –2 0	
5	5	1 1 –1 –1 –1	1 1 –1 –1 –1 1 1 –1 –1 –1 1 1 –1 –1 –1 1 1 –1 –1 –1 1 1 –1 –1 –1	16 (64 %)
			0 0 0 0 0 0 0 2 2 2 0 0 0 2 0 0 –2 0 2 0 –2 0 2 0 2	

- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 2,1}$ and subtract 2 $\boldsymbol{r}_{\Sigma 2,0}$ to get $\boldsymbol{y}_{\Delta 2,2}$ to compute $\boldsymbol{y}_{2,2}$, $\boldsymbol{y}_{2,7}$, $\boldsymbol{y}_{2,12}$, $\boldsymbol{y}_{2,17}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 2,3}$ and subtract 2 $\boldsymbol{r}_{\Sigma 2,2}$ to get $\boldsymbol{y}_{\Delta 2,4}$ to compute $\boldsymbol{y}_{2,4}$, $\boldsymbol{y}_{2,9}$, $\boldsymbol{y}_{2,14}$, $\boldsymbol{y}_{2,19}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 2,0}$ and subtract 2 $\boldsymbol{r}_{\Sigma 2,4}$ to get $\boldsymbol{y}_{\Delta 2,1}$ to compute $\boldsymbol{y}_{2,1}$, $\boldsymbol{y}_{2,6}$, $\boldsymbol{y}_{2,11}$, $\boldsymbol{y}_{2,16}$.
- 10 accesses to add 2 $\boldsymbol{r}_{\Sigma 2,2}$ and subtract 2 $\boldsymbol{r}_{\Sigma 2,1}$ to get $\boldsymbol{y}_{\Delta 2,3}$ to compute $\boldsymbol{y}_{2,3}$, $\boldsymbol{y}_{2,8}$, $\boldsymbol{y}_{2,13}$, $\boldsymbol{y}_{2,18}$.

In software, this corresponds to 24 + 4 × 10 = 64 additions with one memory, or to 20 + 4 + 8 = 32 additions with 5 memories, giving a total of 175 + 32 + 25 = 232 additions (reduction of 62.9 %). In conclusion, with a total of 200 + 65 = 265 accesses, this code requires slightly fewer accesses than code 1.

3) Complexity analysis of codes 3, 4 and 5 :

The codes $\boldsymbol{s}_{z3}$, $\boldsymbol{s}_{z4}$ and $\boldsymbol{s}_{z5}$ contain more nonzero values than the code $\boldsymbol{s}_{z2}$, and the complexity to compute $\boldsymbol{y}_{\Delta 3}$, $\boldsymbol{y}_{\Delta 4}$ or $\boldsymbol{y}_{\Delta 5}$ is the same as $\boldsymbol{y}_{\Delta 2}$. These codes are thus not interesting.

C. Complexity analysis summary

Table A.IV summarizes the number of accesses for all the codes and provides the complexity reduction compared to the traditional implementation. As a reminder, code 1 was already proposed in [6], and Table A.IV shows that this code was indeed better than the traditional method. After the exhaustive search, one code offering slightly better performance has been found, code 2. The number of additions is reduced by 62.9 %, and using one additional memory the hardware implementation can be 17.9 % faster. As for GPS L5, note that the recursion given in [6] is still more efficient for hardware receivers.

VI. Application to the Galileo E5 signal

The problem is a bit different with the Galileo E5 signal, because there is not only one secondary code but one hundred. Therefore, as shown later, it will be difficult to find one pattern that will maximize the number of zeros in $\boldsymbol{s}_z$ for all the codes. Therefore, we will consider that several patterns can be used. Of course, the optimal pattern for each code cannot be considered, because it would make the receiver very complicated. However, few patterns that are similar and correspond to a similar implementation can be considered.

A. Searching of best codes $\boldsymbol{s}_\Delta$ and complexity analysis

1) Pattern of length 1 :

As mentioned previously, for a pattern of length 1, there are two possibilities, +1 or –1. With these two patterns, the number of zeros in $\boldsymbol{s}_z$ is between 45 and 55, as shown in Table VII. Taking into account all the E5 secondary codes, the average number of zeros in $\boldsymbol{s}_z$ is 50.22 with the pattern +1, and 49.78 with the pattern –1. Therefore, if only one pattern is used, the best we can have is 50.22 % of zeros in average (far from the 60 % and 70 % found previously with the E1 and L5 codes). In this case, the implementation is the one shown in Fig. 3, and the average number of accesses would thus be 100 + 49.78 × 99 ≈ 5029 (100 to compute $\boldsymbol{r}_\Sigma$, 49.78 for one output, 99 for all the outputs using the trick mentioned previously where the correlation is computed for the first delay when $\boldsymbol{r}_\Sigma$ is computed).

However, the second pattern is the opposite of the first one, thus Eq. (11) would become

$$\begin{bmatrix} \boldsymbol{y}_{\Delta 1,0} \\ \boldsymbol{y}_{\Delta 1,1} \\ \vdots \\ \boldsymbol{y}_{\Delta 1,N_S-1} \end{bmatrix} = \begin{bmatrix} -1 & -1 & \cdots & -1 \\ -1 & -1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_0 \\ \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_{N_S-1} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{r}_\Sigma \\ -\boldsymbol{r}_\Sigma \\ \vdots \\ -\boldsymbol{r}_\Sigma \end{bmatrix}, \quad (22)$$

and Eq. (12) would become

$$\boldsymbol{y}_k = \boldsymbol{y}_{z1,k} + \boldsymbol{y}_{\Delta 1,k} = \boldsymbol{y}_{z1,k} - \boldsymbol{r}_\Sigma. \quad (23)$$

Thus the implementation with this pattern is similar to Fig. 3 except that $\boldsymbol{r}_\Sigma$ should be subtracted instead of added. Therefore, it is relatively easy to use both patterns, because only the last operation should switch between addition and subtraction depending on the current secondary code. In this case, the most appropriate pattern for each code (the one giving the highest number of zeros in $\boldsymbol{s}_z$) can be used, and doing so the average number of zeros in $\boldsymbol{s}_z$ is 53.44, as shown in Table VII. The average number of accesses would thus be 100 + (100 – 53.44) × 99 ≈ 4710, i.e. a reduction 52.9 % compared to the traditional implementation, or 5.8 % compared to the traditional implementation with an additional memory. This result was already found in [6].

2) Pattern of length 2 :

TABLE VII. Statistics related to $\boldsymbol{s}_z$ with the Galileo E5 signal.

Pattern length	All patterns				Selected patterns				
	Number of patterns	Number of zeros in $\boldsymbol{s}_z$			Number of patterns	Number of zeros in $\boldsymbol{s}_z$			Reduction of the number of accesses (for one additional memory and using the trick)
		Range	Highest mean for one pattern	Mean of the maximums for each code		Range	Highest mean for one pattern	Mean of the maximums for each code	
1	2	45 – 55	50.22	53.44	-	-	-	-	5.80 %
2	2	44 – 56	50.34	53.76	-	-	-	-	6.44 %
4	14	39 – 61	50.60	57.47	4	41 – 59	50.38	56.16	10.18 %
10	1022	35 – 65	51.00	61.90	10	39 – 61	50.44	57.82	12.88 %
20	1 048 574	29 – 71	51.28	68.54	1024	37 – 63	50.48	58.54	14.10 %

For a pattern of length 2, there are two possibilities, +1 –1 or –1 +1. With these two patterns, the number of zeros in $\boldsymbol{s}_z$ is between 44 and 56, as shown in Table VII. Both patterns give an average number of zeros in $\boldsymbol{s}_z$ of 49.66 and 50.34, respectively. However, as for the patterns of length 1, one is the opposite of the other; therefore they can be both implemented easily, using the most appropriate pattern for each code. In this case, the average number of zeros in $\boldsymbol{s}_z$ is 53.76, as shown in Table VII. The average number of accesses would thus be $100 + (100 - 53.76) \times 99 \approx 4678$, i.e. a reduction 53.2 % compared to the traditional implementation, or 6.4 % compared to the traditional implementation with an additional memory. Therefore, these patterns allows a slightly lower processing time than the previous ones.

3) Pattern of length 4 :

For a pattern of length 4, there are 14 possibilities. With these patterns, the number of zeros in $\boldsymbol{s}_z$ is between 39 and 61, as shown in Table VII. This is expected, since there are more patterns it is more likely than some fit better than the only two choices of before. However, as mentioned in the introduction of this section, using only one pattern cannot give good performance because it cannot be good for all the E5 codes. This is shown again in Table VII, where it can be seen the highest average number of zeros using one pattern is 50.60.

If as previously, the best pattern for each code is used, the average number of zeros would be 57.47, which is much higher than previously. However, not every pattern gives the same efficiency in the computation of $\boldsymbol{y}_\Delta$, as it was observed in Sections IV and V and shown in Tables A.III and A.IV. Nevertheless, in these sections, two patterns provided the best efficiency in the computation of $\boldsymbol{y}_\Delta$: codes 4 and 7 of the L5 signal (excluding the pattern of length 1). These patterns had the following properties :

- The second half of the pattern is the opposite of the first half, allowing us to combine the inputs and reducing the size of the matrix because the second half of the output was the opposite of the first half (see Eqs. (16) and (17), and (20)).
- In the skew-circulant matrix obtained (Eqs. (17) and (20)), there is only one difference between two consecutive rows. This is possible only if the first half of the pattern contains first +1 and then –1, or vice versa, but not if +1 and –1 are mixed.

For example the patterns 1 1 1 1 –1 –1 –1 –1 or 1 –1 –1 –1 –1 1 1 1 satisfy these criteria, but not the pattern 1 –1 1 1 –1 1 –1 –1 (a –1 is surrounded by +1 in the first half).

Therefore, among the 14 patterns of length 4, only those satisfying these criteria are selected, which gives 4 patterns (which are in fact the four rows of the matrix in Eq. (16)). These four patterns provide thus efficient computation of $\boldsymbol{y}_\Delta$ (similar to Eqs. (16) and (17)), and it is easy to switch between them because only the order of the rows in the matrix of Eq. (16) changes. In this case, the average number of zeros in $\boldsymbol{s}_z$ is 56.16, as shown in Table VII. The average number of accesses would thus be $(100 + 50) + (100 - 56.16) \times 99 \approx 4491$ ($100 + 50$ is for the computation of $\boldsymbol{y}_\Delta$ using Eq. (17) with $N_S = 100$ instead of 20), i.e. a reduction 55.1 % compared to the traditional implementation, or 10.2 % compared to the traditional implementation with an additional memory. Therefore, these patterns allows a lower processing time than the previous ones.

4) Pattern of length 10 :

For a pattern of length 10, there are 1022 possibilities, whose performance are given in Table VII. As for the pattern of length 4, not all of them can be used; therefore those satisfying the criteria given previously are selected. There are 10 such patterns (with as first half 1 1 1 1 1, then 1 1 1 1 –1, then 1 1 1 –1 –1, etc. until –1 1 1 1 1). In this case, the average number of zeros in $\boldsymbol{s}_z$ is 57.82, as shown in Table VII. The average number of accesses would thus be $(100 + 4 \times 20) + (100 - 57.82) \times 99 \approx 4356$ ($100 + 4 \times 20$ is for the computation of $\boldsymbol{y}_\Delta$ using Eq. (20) with $N_S = 100$ instead of 20), i.e. a reduction 56.4 % compared to the traditional implementation, or 12.9 % compared to the traditional implementation with an additional memory. Therefore, these patterns allows a lower processing time than the previous ones.

5) Pattern of length 20 :

Exactly the same principle apply to patterns of length 20, where there are 1 048 574 possibilities, and only 20 patterns satisfying the criteria given previously. In this case, the average number of zeros in $\boldsymbol{s}_z$ is 58.54, as shown in Table VII. The average number of accesses would thus be $(100 + 9 \times 10) + (100 - 58.54) \times 99 \approx 4295$ (the general formula for the number of accesses for $\boldsymbol{y}_\Delta$ with such patterns is $N_S + (L/2 - 1) \times 2N_S/L = 2N_S(1 - 1/L)$ with L the pattern length), i.e. a reduction 57.1 % compared to the traditional implementation, or 14.1 % compared to the traditional implementation with an additional memory. Therefore, these patterns allows a lower processing time than the previous ones.

B. Summary

With the E5 signals, it is also possible to reduce the complexity. However, in order to obtain interesting results, several patterns should be used. For the simplest case, with only two patterns, the patterns with a length 2 are the best. Then, longer the pattern, better the performance, but more complex the control required to apply the appropriate pattern for each code. However, as for the other signals, the recursion given in [6] is still more efficient for hardware receivers.

VII. CONCLUSION

In this paper, we have focused on a method to decompose the local secondary code in two codes, one with a lot of zeros, and one with a pattern, in order to reduce the complexity of the correlation of the GNSS secondary code, and this without impacting the sensitivity.

This method has already been presented in a previous article, were solution were found by inspection. In this paper, an exhaustive search over all the possibilities has been performed, and the minimum complexity was analyzed. Since the calculations are relatively simple (product of a matrix and a vector with 4 or 5 rows), the complexities found are expected to be the minimal ones; however, there is no theoretical proof.

For all the signals, the complexity is reduced compared to the previous article. Moreover, the paper has presented an

additional trick that reduces slightly the processing time by saving several memory accesses.

For the L5 signal, the processing time for the secondary code correlation for hardware implementations is reduced by 32.5 % compared to 20 % previously (fair comparison with one additional memory), the number of operations can be reduced by 69.3 % compared to 65 %, and the processing time for with Matlab simulation is reduced by about 45 % compared to 41 %.

For the E1 signal, the processing time for the secondary code correlation for hardware implementations is reduced by 17.9 % compared to 12.1 % previously (fair comparison with one additional memory), and the number of operations can be reduced by 62.9 % compared to 56.2 % previously.

For the E5 signal, the processing time for the secondary code correlation for hardware implementations can be reduced by at most 14.1 % compared to 5.8 % previously, and the number of operations can be reduced by 57.1 % compared to 52.9 % previously.

Note that this method can also be applied to secondary code on data channels. Moreover, the BeiDou B1 and B2 secondary code on the data channel is the GPS L5 secondary code, therefore the results provided applies directly to these BeiDou signals.

Finally, note that for a hardware implementation, this method is still less efficient than the recursion presented in [6]. But the recursion has the drawback of adding many values, which requires truncation to keep the resolution reasonable.

REFERENCES

[1] A.J. Van Dierendonck, "New GNSS signals: Will modern also be better?", Inside GNSS, vol. 9, no. 2, pp. 42-43, March/April 2014.

[2] L. Lo Presti, X. Zhu, M. Fantino, P. Mulassano, "GNSS signal acquisition in the presence of sign transition", IEEE Journal of Selected Topics in Signal Processing, vol. 3, no. 4, pp. 557-570, 2009.

[3] D. Borio, M. Fantino, L. Lo Presti, "The impact of the Galileo signal in space in the acquisition system," in Satellite Communications and Navigation Systems, Springer US, 2008.

[4] C. Yang, C. Hegarty, M. Tran, "Acquisition of the GPS L5 signal using coherent combining of I5 and Q5", *ION GNSS*, pp. 2184-2195, Sept. 2004.

[5] J. Betz, "Engineering Satellite-Based Navigation and Timing: Global Navigation Satellite Systems, Signals, and Receivers", Wiley-IEEE Press, 2016.

[6] J. Leclère, R. Jr, Landry, "Complexity reduction for high sensitivity acquisition of GNSS signals with a secondary code", ION GNSS+ 2016, Portland, Oregon, USA, September 2016, pp. 436-443.

[7] IS-GPS-705 Revision D, Navstar GPS Space Segment/User Segment L5 Interfaces, Sept. 2013.

[8] Galileo SIS ICD Issue 1.2, European GNSS (Galileo) Open Service, Nov. 2015.

[9] IS-GPS-800 Revision D, Navstar GPS Space Segment/User Segment L1C Interfaces, Sept. 2013.

[10] H. Mathis, P. Flammant, A. Thiel, "An analytic way to optimize the detector of a postcorrelation FFT acquisition algorithm", *ION GPS/GNSS*, pp. 689-699, Sept. 2003.

[11] J. Leclère, "Resource-efficient parallel acquisition architectures for modernized GNSS signals", Ph.D. thesis, EPFL, Switzerland, 2014.

[12] N.I. Ziedan, J.L. Garrison, "Unaided acquisition of weak GPS signals using circular correlation or double-block zero padding", *IEEE PLANS*, April 2004, pp. 461–470.

[13] M. Foucras, O. Julien, C. Macabiau, B. Ekambi, "A novel computationally efficient Galileo E1 OS acquisition method for GNSS software receiver", *ION GNSS*, pp. 365-383, Sept. 2012.

[14] D. Akopian, Fast FFT based GPS satellite acquisition methods, *IEE Proceedings Radar, Sonar and Navigation*, vol. 152, no. 4, pp. 277-286, August 2005.

[15] J. Leclère, C. Botteron, R. Jr Landry, P.-A. Farine, "FFT Splitting for Improved FPGA-Based Acquisition of GNSS Signals", International Journal of Navigation and Observation, pp. 1-12, vol. 2015.

[16] J. Leclère, C. Botteron, P.-A. Farine, "High sensitivity acquisition of GNSS signals with secondary code on FPGAs", IEEE Aerospace and Electronic Systems Magazine, accepted.

APPENDIX

TABLE A.I. PATTERNS OF LENGTH 10 PROVIDING 14 ZEROS IN s_z WITH THE GPS L5 SIGNAL.

1 1 1 1 1 1 1 1 -1 1	1 1 1 –1 1 1 1 1 –1 1	1 –1 1 1 1 1 1 1 –1 1	1 –1 1 –1 1 1 1 1 –1 1
1 1 1 1 1 1 1 1 –1 –1	1 1 1 –1 1 1 1 1 –1 –1	1 –1 1 1 1 1 1 1 –1 –1	1 –1 1 –1 1 1 1 1 –1 –1
1 1 1 1 1 1 1 –1 –1 1	1 1 1 –1 1 1 1 –1 –1 1	1 –1 1 1 1 1 1 –1 –1 1	1 –1 1 –1 1 1 1 –1 –1 1
1 1 1 1 1 1 1 –1 –1 –1	1 1 1 –1 1 1 1 –1 –1 –1	1 –1 1 1 1 1 1 –1 –1 –1	1 –1 1 –1 1 1 1 –1 –1 –1
1 1 1 1 1 1 –1 1 –1 1	1 1 1 –1 1 1 –1 1 –1 1	1 –1 1 1 1 1 –1 1 –1 1	1 –1 1 –1 1 1 –1 1 –1 1
1 1 1 1 1 1 –1 1 –1 –1	1 1 1 –1 1 1 –1 1 –1 –1	1 –1 1 1 1 1 –1 1 –1 –1	1 –1 1 –1 1 1 –1 1 –1 –1
1 1 1 1 1 1 –1 –1 –1 1	1 1 1 –1 1 1 –1 –1 –1 1	1 –1 1 1 1 1 –1 –1 –1 1	1 –1 1 –1 1 1 –1 –1 –1 1
1 1 1 1 1 1 –1 –1 –1 –1	1 1 1 –1 1 1 –1 –1 –1 –1	1 –1 1 1 1 1 –1 –1 –1 –1	1 –1 1 –1 1 1 –1 –1 –1 –1
1 1 1 1 1 –1 1 1 –1 1	1 1 1 –1 1 –1 1 1 –1 1	1 –1 1 1 1 –1 1 1 –1 1	1 –1 1 –1 1 –1 1 1 –1 1
1 1 1 1 1 –1 1 1 –1 –1	1 1 1 –1 1 –1 1 1 –1 –1	1 –1 1 1 1 –1 1 1 –1 –1	1 –1 1 –1 1 –1 1 1 –1 –1
1 1 1 1 1 –1 1 –1 –1 1	1 1 1 –1 1 –1 1 –1 –1 1	1 –1 1 1 1 –1 1 –1 –1 1	1 –1 1 –1 1 –1 1 –1 –1 1
1 1 1 1 1 –1 1 –1 –1 –1	1 1 1 –1 1 –1 1 –1 –1 –1	1 –1 1 1 1 –1 1 –1 –1 –1	1 –1 1 –1 1 –1 1 –1 –1 –1
1 1 1 1 1 –1 –1 1 –1 1	1 1 1 –1 1 –1 –1 1 –1 1	1 –1 1 1 1 –1 –1 1 –1 1	1 –1 1 –1 1 –1 –1 1 –1 1
1 1 1 1 1 –1 –1 1 –1 –1	1 1 1 –1 1 –1 –1 1 –1 –1	1 –1 1 1 1 –1 –1 1 –1 –1	1 –1 1 –1 1 –1 –1 1 –1 –1
1 1 1 1 1 –1 –1 –1 –1 1	1 1 1 –1 1 –1 –1 –1 –1 1	1 –1 1 1 1 –1 –1 –1 –1 1	1 –1 1 –1 1 –1 –1 –1 –1 1
1 1 1 1 1 –1 –1 –1 –1 –1	1 1 1 –1 1 –1 –1 –1 –1 –1	1 –1 1 1 1 –1 –1 –1 –1 –1	1 –1 1 –1 1 –1 –1 –1 –1 –1

TABLE A.II. RECURSION WITH THE GPS L5 SIGNAL.

Code ID	Number of nonzero values in s_z	Recursion : Number of differences between $s_{z,i-k}$ and $s_{z,i}$																		Number of accesses to compute y_z
		k																		Good ?
		1	**2**	**3**	**4**	**5**	**9**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	
1	8	10	10	10	10	10	12	10	8	10	12	10	8	10	12	10	10	10	10	10	×	8 × 20 = 160
2	8	13	10	13	10	13	12	13	8	13	12	13	8	13	12	13	10	13	10	13	×	8 × 20 = 160
3	5	10	8	7	10	7	10	7	8	10	4	10	8	7	10	7	10	7	8	10	✓	(5 + 4) × 10 = 90
4	6	9	11	9	10	12	10	9	8	9	10	9	8	9	10	12	10	9	11	9	×	6 × 20 = 120
5	6	10	9	12	10	10	6	9	7	10	12	10	7	9	6	10	10	12	9	10	×	6 × 20 = 120
6	6	11	9	9	8	10	10	12	7	11	12	11	7	12	10	10	8	9	9	11	×	6 × 20 = 120
7	6	10	9	12	8	11	10	9	7	10	12	10	7	9	10	11	8	12	9	10	×	6 × 20 = 120

TABLE A.III. SUMMARY OF THE NUMBER OF ACCESSES WITH THE GPS L5 SIGNAL. THE ADDITIONAL TRICK CONSISTS IN COMPUTING THE FIRST $y_{z,k}$ WHILE ACCESSING THE INPUTS TO COMPUTE y_Δ OR INTERMEDIATE RESULTS RELATED TO y_Δ.

Code	Additional memories	Accesses for y_z	Accesses for y_Δ	Total	Reduction compared to the traditional implementation		Total with the additional trick	Reduction with the additional trick	
					without additional memory	with the same number of additional memory		without additional memory	with the same number of additional. memory
Traditional	0	-	-	400	-	-	400	-	-
	1	-	-	200	50 %	-	200	50 %	-
	2	-	-	134	66.5 %	-	134	66.5 %	-
	3	-	-	100	75 %	-	100	75 %	-
1, 2	1	160	20	180	55 %	10 %	172	57 %	14 %
3	1	90	50	140	65 %	30 %	135	66.3 %	32.5 %
	2		35	125	68.8 %	6.7 %	120	70 %	10.4 %
	3		20	110	72.5 %	–10 %	105	73.8 %	–5 %
4	1	120	30	150	62.5 %	25 %	144	64 %	28 %
	2		20	140	65 %	–4.5 %	134	66.5 %	0 %
5, 6	1	120	52	172	57 %	14 %	166	58.5 %	17 %
7	1	120	36	156	61 %	22 %	150	62.5 %	25 %

TABLE A.IV. SUMMARY OF THE COMPLEXITY WITH THE GALILEO E1 SIGNAL.

Code	Additional memories	Accesses for y_z	Accesses for y_Δ	Total	Reduction compared to the traditional implementation		Total with the additional trick	Reduction with the additional trick	
					no additional memory	same number of additional memory		no additional memory	same number of additional. memory
Traditional	0	-	-	625	-	-	625	-	-
	1	-	-	313	49.9 %	-	313	49.9 %	-
	2	-	-	209	66.6 %	-	209	66.6 %	-
	3	-	-	157	74.9 %	-	157	74.9 %	-
1	1	250	25	275	56 %	12.1 %	265	57.6 %	15.3 %
2	1	200	65	265	57.6 %	15.3 %	257	58.9 %	17.9 %
3, 4, 5	1	225	65	290	53.6 %	7.3 %	281	55 %	10.2 %