Presented at ION GNSS 2016, Session A4, Portland, OR, USA, September 12 - 16, 2016.

Complexity reduction for high sensitivity acquisition of GNSS signals with a secondary code

Jérôme Leclère, René Jr Landry, *LASSENA, ÉTS, Montréal, Canada*

BIOGRAPHIES

Jérôme Leclère received an engineering degree in Electronics and Signal Processing from ENSEEIHT, Toulouse, France, in 2008, and his Ph.D. in the GNSS field from EPFL, Switzerland, in 2014. He is now postdoctoral researcher at LASSENA, ÉTS, Montreal, Canada. His research focuses on the complexity reduction of GNSS signals acquisition techniques, and on GNSS/INS integration for automotive applications.

René Jr Landry received an electrical engineering degree at École Polytechnique of Montreal in 1992, a Master of Science in Satellite Communication engineering in 1993, a DEA in microwave in 1994 and a Ph.D. in GNSS anti-jamming at SupAero in 1998. Since 1999, he is professor at the department of electrical engineering at École de Technologie Supérieure (ÉTS), and the director of LASSENA laboratory. His expertise in embedded systems, navigation and avionic is applied notably in the field of transport, aeronautic, and space technologies.

ABSTRACT

Modern GNSS signals include several innovations compared to the GPS L1 C/A signal. Among them, there is the secondary code and the pilot channel that is free of data. The use of a secondary code brings several advantages, but it also complicates the acquisition process. Indeed, it implies a potential sign transition between each consecutive primary code period, which is problematic when dealing with circular correlation. This paper focuses on high sensitivity acquisition with fast Fourier transform (FFT) based parallel code search, to propose three different methods to reduce the complexity of the correlation with the secondary code, without impacting the detection performance. The idea behind the methods is to make zeros appears in the secondary code. The proposed methods are applied to the GPS L5, and Galileo E1 OS and E5 signals, and it is shown that the number of operations and the processing time for both software and hardware receiver can always be halved for any code with the first method. The two other methods provide even better performance, with a reduction up to 76 %, although not applicable to every code.

1. INTRODUCTION

Modern GNSS signals include a secondary code to create what is called a tiered code, and the addition of a pilot channel that is free of data. The pilot channel presents some clear advantages, such as allowing longer coherent integration times (necessary to have high sensitivity [1]) and the use of better discriminators in the tracking loops, which can significantly increase the receiver sensitivity and robustness. The secondary code also brings several advantages, for example it makes the data synchronization process easier; it can improve the cross-correlation between different satellites if each one has its own secondary code; it improves the interference mitigation; and it allows basic receivers to still acquire and process the GNSS signal using only the primary code [2-3]. However, the secondary code complicates the acquisition process. Indeed, now there is a potential sign transition between each consecutive primary code period, which is problematic when dealing with circular correlation [4].

This paper focuses on the acquisition of GNSS signals with the secondary code synchronization. Starting from the well-known FFT based parallel code search (PCS) [5-6], several additional algorithms to reduce the complexity will be presented and compared. In fact, these algorithms can be applied with other methods, the PCS being considered simply because it can be easily formulated mathematically. The complexity reduction refers to a reduction of the theoretical number of operations, and to a reduction of the processing time for both software and hardware based receivers. The algorithms proposed are not approximations; they are simply different ways of doing a calculation, therefore the detection performance are not affected.

To reach this goal, we started to look at the problem using different mathematical tools, namely expressing the signals and correlation operation in the time domain, the frequency domain, using the z-transform, and using matrix notation. The matrix notation was the most helpful and therefore this is the one presented and used here to show the proposed algorithms (some other mathematical tools are still shown in Appendix).

The next section introduces the model of the received signal and the tiered code, and presents some characteris-

tics of several GNSS secondary codes. Section 3 shows the acquisition architecture, and discusses the computation of the correlation with the secondary code for pilot channels. Then, Section 4 presents different methods to reduce the complexity of the secondary code correlation.

2. GNSS SIGNALS MODEL

2.1 Received signal

The signal received by a GNSS receiver is the combination of several GNSS signals coming from U different satellites plus a noise term. Thus, after the front-end, the discrete baseband signal can be written as

$$s_b(nT_S) = \sum_{u=1}^{U} s_b^u(nT_S) + \eta_b(nT_S), \quad \textbf{(1)}$$

where s_b^u is the discrete baseband signal from satellite u, n is the discrete time index, T_S is the sampling interval equal to $1/f_S$ with f_S being the sampling frequency, and η_b is the noise component.

Considering a real sampling front-end, the discrete baseband signal from satellite u can be expressed as

$$\begin{aligned} s_b^u(nT_S) = &\sqrt{2P_d^u}\, d^u(nT_S - \tau^u) c_d^u(nT_S - \tau^u) \\ &\cos\left(2\pi f_b^u n T_S + \varphi_{b,d}^u\right) \\ &+ \sqrt{2P_p^u}\, c_p^u(nT_S - \tau^u) \sin\left(2\pi f_b^u n T_S + \varphi_{b,p}^u\right), \end{aligned} \quad \textbf{(2)}$$

where P_d^u and P_p^u are the powers of the data and pilot channel respectively, c_d^u and c_p^u are sequences composed of a primary code (also called spreading code or pseudo random noise (PRN) code) and possibly a secondary code (also called overlay code) and a subcarrier (not considered here but without impact for our discussion) for the data and pilot channels respectively, d^u is the data sequence, τ^u is an unknown delay, f_b^u is the baseband frequency that includes the intermediate frequency, and a residual frequency due to the Doppler effect and the local oscillator, and $\varphi_{b,d}^u$ and $\varphi_{b,p}^u$ are the carriers phases [6-7].

2.2 Tiered code

2.2.1 Tiered code model

A code composed of a primary and a secondary code is called a tiered code. In a tiered code, the primary code is repeated several times and each period is multiplied by a chip of the secondary code. Since both primary and secondary codes are binary, the tiered code is also binary.

We denote N_P the length of the primary code in samples, N_S the length of the secondary code in chip, thus $N = N_P N_S$ is the length of the tiered code in samples. Generally $N_P \gg N_S$. For the following, we denote $c[n]$ or c_n the tiered code with $n = 0, 1, \ldots, N-1$; $p[n_p]$ or p_{n_p} the primary code with $n_p = 0, 1, \ldots, N_P - 1$; and $s[n_s]$ or s_{n_s} the secondary code with $n_s = 0, 1, \ldots, N_S - 1$.

$p[n_p]$	p_0	p_1	p_2	p_3	p_0	p_1	p_2	p_3	p_0	p_1	p_2	p_3
×												
$s[n_s]$	s_0	s_0	s_0	s_0	s_1	s_1	s_1	s_1	s_2	s_2	s_2	s_2
=												
$c[n]$	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}

Figure 1 : Illustration of a tiered code with $N_P = 4$ $N_S = 3$, $N = 12$.

Table 1 : Value of n in function of n_p and n_s with $N_P = 4$, $N_S = 3$, $N = 12$.

n_s \ n_p	**0**	**1**	**2**	**3**
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

Figure 1 illustrates a tiered code and the relation between $p[n_p]$, $s[n_s]$ and $c[n]$, and Table 1 shows the relation between n, n_p and n_s. From this, it can be seen that $n = n_p + N_P n_s$ and $c[n] = p[n_p]s[n_s]$.

Using matrix notation, denoting **c** the tiered code, **p** the primary code and **s** the secondary code, they can be expressed as

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N_P-1} \end{bmatrix}, \quad \mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{N_S-1} \end{bmatrix}, \quad \textbf{(3)}$$

and they can be related using the Kronecker product as

$$\mathbf{c} = \mathbf{s} \otimes \mathbf{p} = \begin{bmatrix} s_0\mathbf{p} \\ s_1\mathbf{p} \\ \vdots \\ s_{N_S-1}\mathbf{p} \end{bmatrix}. \quad \textbf{(4)}$$

As for the z-transform and discrete Fourier transform (DFT) of a tiered code, they can be found in Appendix.

2.2.2 Secondary codes characteristics

For a GNSS signal, the secondary code is the same for the different satellites, except for the L1C and E5 signals where each satellite has its own secondary code. Table 2 shows some characteristics of the pilot channel secondary codes, which will be used in Section 4.

Table 2 : Pilot channel secondary codes characteristics.

Signal	Length (chip)	Number of 1	Autocorrelation side peaks
L5	20	12 (60 %)	0, ±4
E1	25	15 (60 %)	1, –3
E5a/b	100	45 to 55	0, ±4, ±8

3. GNSS SIGNAL ACQUISITION

The goal of the acquisition is to detect a GNSS signal and coarsely estimate its code delay τ^u and baseband frequency f_b^u. For this, the baseband frequency should be removed and a correlation with the code should be performed. There are many ways to do this, such as a serial search [5], a parallel frequency search [6-8], a parallel code search (PCS) [6], or two-dimensional search [7,9-11]. Here, we consider the PCS where the correlation is performed on the primary code period (doubling the length of the sequences and using zero-padding to manage the sign transition), repeating this for several periods, and then another correlation is performed on the secondary code, as suggested in [12]. With this method, it is not expected to detect the signal after the correlation on the primary code as does a two steps acquisition [13], therefore high sensitivity is possible.

The corresponding schematic is given in Figure 2. The first correlation is circular and performed by FFTs for the sake of efficiency. The FFT of the local code can be computed only once and stored, as long as the Doppler effect on the code is taken into account. Due to the potential sign transition, the length of the sequence is doubled, i.e. the equivalent of two periods is used for the incoming signal and the local primary code is padded with N_P zeros [14-16], else there would be a strong impact on the probability of detection [17]. More zeros can be padded if a specific length must be reached (a power of two, for example). At the output, only the first N_P samples are kept because the others may suffer a loss in case of sign transition. The second correlation with the secondary code can be or not circular, since there is no additional transition with the pilot channel. The advantage of the circular correlation is that it requires less input data, therefore it is the one considered thereafter. Then, all this can be done for several tiered code periods and the results can be accumulated coherently or non-coherently.

In summary, the incoming signal of length of a tiered code period is multiplied by a local carrier, and then it is separated into N_S blocks denoted $\mathbf{x}_i$ of $2N_P$ samples (the blocks overlap). Each block is used to perform a circular correlation with the local zero-padded primary code $\mathbf{p}_z$, giving N_S correlation results of length N_P, denoted $\mathbf{r}_i$. Then the correlation results $\mathbf{r}_i$ are correlated with the local secondary code $\mathbf{s}$ as follows

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

where $\mathbf{S}$ is a circulant matrix with $\mathbf{s}^{\mathrm{T}}$ as first row [18]. As shown in Figure 2, Eq. (5) can also be written as

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{i-k}\,\mathbf{r}_i, \tag{6}$$

where the subscript of s is modulo N_S (the same applies for the next equations when a subscript may be lower than 0 of higher than $N_S - 1$).

The circular correlation in Eq. (5) could be computed using FFTs, however it may not be efficient [19]. Firstly, except for the E5 signal, the secondary codes are rather short (20 or 25 chips), and FFT-based correlation are not efficient for such short lengths [20], especially if zero-padding is necessary. Secondly, the secondary codes are binary, therefore no actual multiplications are needed, the correlation can be computed only with additions and subtractions based on the value of the code. Thus, it is not a good idea to involve the FFTs complex exponentials (this last argument is valid because the secondary codes are short; it is not valid for codes of thousands of chips). Therefore, each row in Eq. (5) requires $N_S - 1$ additions (subtractions are counted as additions). There are N_S rows, thus there are $N_S(N_S - 1)$ additions to perform altogether, i.e. 380, 600 and 9900 additions for the L5, E1 and E5 signals, respectively.

For a hardware implementation, the processing time is directly proportional to the number of accesses to $\mathbf{r}_i$ [19]. There are N_S accesses to compute each $\mathbf{y}_k$, thus the total number of accesses is N_S^2.

In Figure 2, the secondary code correlation counts for about 25 % of all the operations, but for a hardware implementation it can count for more than 90 % of the processing time [19]. Therefore, optimizing the secondary code correlation is essential. Optimizations of the primary code correlation can be found in [15-16], and Section 4 provides this for the secondary code correlation.

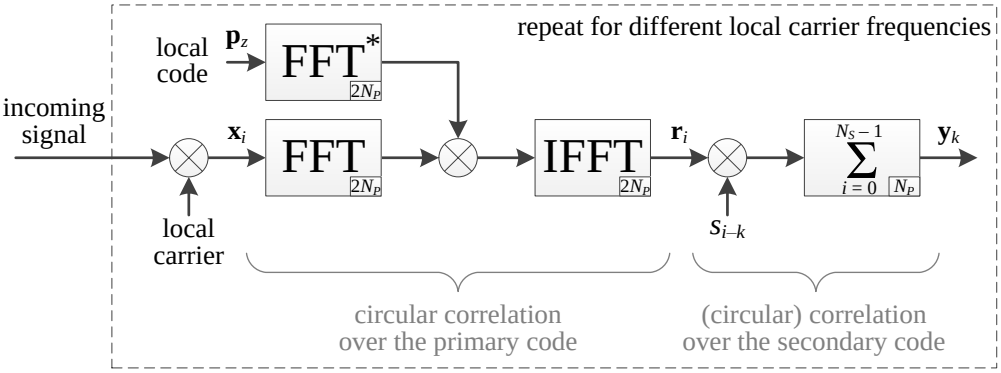

Figure 2 : Acquisition architecture considered.

4. EFFICIENT SECONDARY CODE CORRELATIONS

There are already many fast algorithms for convolution and correlation, whose purpose is primarily to reduce the number of multiplications at the expense of supplementary additions [21-24]. However, as mentioned previously, the correlation of the secondary codes involves only additions since these codes are binary and short. Thus, such fast algorithms are not useful here, and it is necessary to find new strategies to reduce the complexity.

The selected strategy is to make zeros appear in the matrix of the secondary code, because each zero means one addition less. This section presents three methods to do this, and for each one, the number of additions and the hardware processing time are evaluated, and the reduction of the processing time obtained with Matlab simulations is given. For the Matlab simulations, the processing time has been measured over 200 iterations of the algorithms, and 100 measures have been performed and averaged.

4.1 Method #1 : Adding ±1 to the code

The first method consists in writing the secondary code as the sum of a code and ±1, e.g. $\mathbf{s} = \mathbf{s} - \mathbf{1}_{N_S\times 1} + \mathbf{1}_{N_S\times 1} = \mathbf{s}_1 + \mathbf{1}_{N_S\times 1}$, where $\mathbf{1}_{N_S\times 1}$ is a vector of length N_S containing only ones, and the chips of $\mathbf{s}_1 = \mathbf{s} - \mathbf{1}_{N_S\times 1}$ can take as value 0 or -2. Using this in Eq. (5) gives

$$\begin{aligned}\mathbf{y} &= \mathbf{S}\,\mathbf{r}\\ \mathbf{y} &= \left(\mathbf{S}_1 + \mathbf{1}_{N_S\times N_S}\right)\mathbf{r}\\ \mathbf{y} &= \mathbf{S}_1\,\mathbf{r} + \mathbf{1}_{N_S\times N_S}\,\mathbf{r}\end{aligned}$$

$$\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \vdots\\ \mathbf{y}_{N_S-1}\end{bmatrix} = \begin{bmatrix} s_{1,0} & s_{1,1} & \cdots & s_{1,N_S-1}\\ s_{1,N_S-1} & s_{1,0} & \cdots & s_{1,N_S-2}\\ \vdots & \vdots & \ddots & \vdots\\ s_{1,1} & s_{1,2} & \cdots & s_{1,0}\end{bmatrix}\begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \vdots\\ \mathbf{r}_{N_S-1}\end{bmatrix} + \begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & \cdots & 1\end{bmatrix}\begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \vdots\\ \mathbf{r}_{N_S-1}\end{bmatrix} \quad \textbf{(7)}$$

$$\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \vdots\\ \mathbf{y}_{N_S-1}\end{bmatrix} = \begin{bmatrix} s_{1,0} & s_{1,1} & \cdots & s_{1,N_S-1}\\ s_{1,N_S-1} & s_{1,0} & \cdots & s_{1,N_S-2}\\ \vdots & \vdots & \ddots & \vdots\\ s_{1,1} & s_{1,2} & \cdots & s_{1,0}\end{bmatrix}\begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \vdots\\ \mathbf{r}_{N_S-1}\end{bmatrix} + \begin{bmatrix}\mathbf{r}_\Sigma\\ \mathbf{r}_\Sigma\\ \vdots\\ \mathbf{r}_\Sigma\end{bmatrix},$$

where $\mathbf{S}_1$ is a circulant matrix with $\mathbf{s}_1^{\mathrm{T}}$ as first row, and $\mathbf{r}_\Sigma = \mathbf{r}_0 + \mathbf{r}_1 + \cdots + \mathbf{r}_{N_S-1}$. Now, the matrix $\mathbf{S}_1$ contains at least half of zeros, at the expense of the additional computation of $\mathbf{r}_\Sigma$. Eq. (7) can also be written as

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{1,i-k}\,\mathbf{r}_i + \mathbf{r}_\Sigma. \quad \textbf{(8)}$$

Figure 3 shows the implementation of Eq. (8).

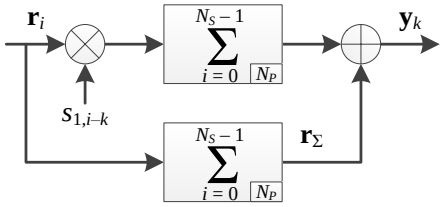

Figure 3 : Implementation of the method #1.

Computing $\mathbf{r}_\Sigma$ requires $N_S - 1$ additions. Denoting R the ratio between the number of non-zero values in the modified local code ($\mathbf{s}_1$) and N_S, the number of non-zero values in $\mathbf{s}_1$ is RN_S. Thus, for each row, there are $RN_S - 1$ additions for the product between $\mathbf{S}_1$ and $\mathbf{r}$, and one addition to add $\mathbf{r}_\Sigma$, i.e. RN_S additions. Therefore, there are $RN_S^2 + N_S - 1$ additions to perform altogether.

The L5 secondary code contains 12 "+1" and 8 "–1" (see Table 3), so subtracting one is the best option to have as many zeros as possible, which gives $R = 0.4$ and 179 additions to perform, i.e. a reduction of 52.9 % compared to the traditional computation. For E1, $R = 0.4$ also and there are 274 additions for a reduction of 54.3 %; and for E5, in the best case $R = 0.45$ which gives 4599 additions for a reduction of 53.5 %, and in average $R = 0.4656$ which gives 4757 additions for a reduction of 51.9 %.

Matlab simulations show a reduction of the processing time of about 39.5 % with the L5 signal, as shown by Figure 6, 47.0 % with the E1 signal, and 49.2 % with the E5 signal using one of the best codes (one having a maximum of "+1").

For a hardware implementation, the reduction of the processing time depends on the number of accesses to $\mathbf{r}_i$. There are firstly N_S accesses to compute $\mathbf{r}_\Sigma$, then there are RN_S accesses to compute each $\mathbf{y}_k$, which gives a total of $RN_S^2 + N_S$ accesses, compared to N_S^2 for the traditional implementation. The reduction is thus equal to $1 - (R + 1/N_S)$. Therefore, compared to the traditional implementation, the processing time is reduced by 55 % for the L5 signal, by 56 % for the E1 signal, and by 52 % in average for the E5 signal. However, the implementation requires an additional memory to store $\mathbf{r}_\Sigma$ [19].

4.2 Method #2 : Adding a code to the code

The first method simply adds or subtracts 1 to the secondary code. But it is also possible to add or subtract another code that would be easy to correlate with, i.e. we can write $\mathbf{s} = \mathbf{s} - \mathbf{s}_\Delta + \mathbf{s}_\Delta = \mathbf{s}_2 + \mathbf{s}_\Delta$, where $\mathbf{s}_\Delta$ can take as value ± 1, and $\mathbf{s}_2$ can take as value 0 or ± 2. Applying this in Eq. (5) gives

$$\begin{aligned}\mathbf{y} &= \mathbf{S}\,\mathbf{r}\\ \mathbf{y} &= (\mathbf{S}_2 + \mathbf{S}_\Delta)\,\mathbf{r}\\ \mathbf{y} &= \mathbf{S}_2\,\mathbf{r} + \mathbf{S}_\Delta\,\mathbf{r},\end{aligned} \quad \textbf{(9)}$$

where $\mathbf{S}_2$ and $\mathbf{S}_\Delta$ are circulant circular matrices with $\mathbf{s}_2^{\mathrm{T}}$ and $\mathbf{s}_\Delta^{\mathrm{T}}$ as first row respectively. The idea is still to have as many zeros as possible in the matrix $\mathbf{S}_2$, and to have $\mathbf{s}_\Delta$ in order that $\mathbf{S}_\Delta\,\mathbf{r}$ can be computed with as few operations

Table 3 : L5 secondary code expressed as the sum of two codes.

$\mathbf{s}$	1	1	1	1	1	–1	1	1	–1	–1	1	–1	1	–1	1	1	–1	–1	–1	1
=																				
$\mathbf{s}_2$	2	2	0	0	2	0	0	0	0	0	0	–2	2	0	0	0	0	0	–2	0
+																				
$\mathbf{s}_\Delta$	–1	–1	1	1	–1	–1	1	1	–1	–1	1	1	–1	–1	1	1	–1	–1	1	1

as possible. For this, typically $\mathbf{s}_\Delta$ should contain a small sequence repeated several times.

Such a code $\mathbf{s}_\Delta$ has been found for the L5 secondary code, which allows having only 6 non-zero values in $\mathbf{s}_2$ (compared to 8 in $\mathbf{s}_1$ with method 1), as shown in Table 3. With this code, only $5 \times 20 = 100$ additions are required to compute $\mathbf{S}_2\,\mathbf{r}$, compared to $7 \times 20 = 140$ additions for $\mathbf{S}_1\,\mathbf{r}$. Now, we will show that $\mathbf{S}_\Delta\,\mathbf{r}$ can be computed efficiently. Expanding the matrix $\mathbf{S}_\Delta$ in Eq. (9), we have

$$\mathbf{S}_\Delta\,\mathbf{r} = \begin{bmatrix} -1 & -1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & -1 & -1 & 1 & \cdots & -1 & 1 \\ 1 & 1 & -1 & -1 & \cdots & -1 & -1 \\ -1 & 1 & 1 & -1 & \cdots & 1 & -1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & 1 & 1 & -1 & \cdots & 1 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \vdots \\ \mathbf{r}_{18} \\ \mathbf{r}_{19} \end{bmatrix}. \quad \textbf{(10)}$$

The third row is the opposite of the first one, the fourth the opposite of the second, and then the rows repeat. Thus, only the first two rows need to be computed. The direct computation of the first two rows of Eq. (10) would require 19 additions per row, i.e. 38 additions in all. However, the pattern $\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ repeats in the matrix $\mathbf{S}_\Delta$, thus taking into account the first two rows only, Eq. (10) can be rewritten as

$$\begin{bmatrix} \mathbf{r}_{\Delta,0} \\ \mathbf{r}_{\Delta,1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \left(\begin{bmatrix} \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{r}_6 \\ \mathbf{r}_7 \end{bmatrix} + \cdots + \begin{bmatrix} \mathbf{r}_{18} \\ \mathbf{r}_{19} \end{bmatrix} - \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_1 \end{bmatrix} - \begin{bmatrix} \mathbf{r}_4 \\ \mathbf{r}_5 \end{bmatrix} - \cdots - \begin{bmatrix} \mathbf{r}_{16} \\ \mathbf{r}_{17} \end{bmatrix} \right), \quad \textbf{(11)}$$

which requires only $2 + 2 \times 9 = 20$ additions. Thus, Eq. (9) can then also be expressed as

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{2,i-k}\,\mathbf{r}_i \pm \mathbf{r}_{\Delta,0}, \quad \textbf{(12)}$$

when k is even, and as

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{2,i-k}\,\mathbf{r}_i \pm \mathbf{r}_{\Delta,1}, \quad \textbf{(13)}$$

when k is odd. Figure 4 shows the corresponding implementation. Therefore, the total number of additions is $(RN_S - 1)N_S + 2N_S = RN_S^2 + N_S$, with R being the ratio between the number of non-zero values in s_2 and N_S. This is very similar to method #1, the difference being in the value of R. There are thus 140 additions for the L5 signal, i.e. 63.1 % less than the traditional computation and 21.8 % less than with method #1.

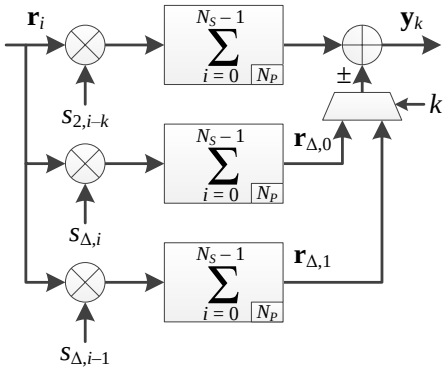

Figure 4 : Implementation of the method #2.

This method can be applied to the E5 signal, with $\mathbf{s}_\Delta$ having the same pattern and where $\mathbf{s}_2$ would contain between 41 and 49 non-zero values according to the code (43.84 on average), compared to 45 with method #1. This would lead to a reduction of 57.5 % at most and 54.7 % in average compared to the traditional implementation, which means an improvement of 8.7 % at most compared to method #1, and of 2.5 % on average, which is not that significant. Regarding the E1 signal, this method cannot be applied because its length is odd, which prevents to find an efficient code $\mathbf{s}_\Delta$.

Matlab simulations show a reduction of the processing time of 41.6 % with the L5 signal, as shown in Figure 6. This is less than foreseen by the number of additions, but a bit greater than the reduction obtained with method #1.

Regarding the hardware implementation, N_S accesses are needed to compute Eq. (11), and there are RN_S accesses to compute each $\mathbf{y}_k$, which gives a total of $RN_S^2 + N_S$ accesses, as with method #1, the difference being in the value of R. For the L5 signal, $R = 6/20 = 0.3$, thus the processing time is reduced by 65 % compared to the traditional implementation. As for the E5 signal, the reduction will be 55.2 % in average. However, the implementation requires two additional memories to store the results of Eq. (11). But it would be possible to have only one additional memory by computing first $\mathbf{y}_{2k}$ (which needs only $r_{\Delta,0}$), and then $\mathbf{y}_{2k+1}$ (which needs only $r_{\Delta,1}$), at the expense of N_S additional accesses. This would give a reduction of the processing time of 60 % for L5, which is still be better than with method #1.

4.3 Method #3 : Recursive computation of correlation

The third method consists in computing the correlation for an initial delay, e.g. $\mathbf{y}_0$, and then computing the correlation for the other delays based on a previously computed result, e.g. compute $\mathbf{y}_1$ from $\mathbf{y}_0$, $\mathbf{y}_2$ from $\mathbf{y}_1$, etc. Indeed, starting from Eq. (6), we can express $\mathbf{y}_k$ from $\mathbf{y}_{k-m}$ as

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{i-k}\,\mathbf{r}_i - \mathbf{y}_{k-m} + \mathbf{y}_{k-m}$$

$$\mathbf{y}_k = \sum_{i=0}^{N_S-1} s_{i-k}\,\mathbf{r}_i - \sum_{i=0}^{N_S-1} s_{i-(k-m)}\,\mathbf{r}_i + \mathbf{y}_{k-m} = \sum_{i=0}^{N_S-1} s_{r_m,i-k}\,\mathbf{r}_i + \mathbf{y}_{k-m}, \quad (14)$$

where $s_{r_m,i-k} = s_{i-k} - s_{i+m-k}$, i.e. $s_{r_m,i} = s_i - s_{i+m}$. The goal is then that $s_{r,i-k}$ contains as many zeros as possible.

This idea may look counterintuitive at the first glance because the codes autocorrelation is very low, which means that when a code is delayed, it should become a quite different code. Thus, if s_i and s_{i+m} are quite different, $s_{r_m,i}$ will not contain a lot of zeros. However, it will be shown it that it may be the case.

Starting with the L5 secondary code, Table 4 gives the number of chips different between the code and its delayed versions, i.e. the number of non-zero values in s_{r_m} according to m. For example, if we consider the code and the same code delayed by one chip (this shift is circular, i.e. the last chip comes at the beginning), they have 10 common and 10 different chips. It can be seen that the number of different chips is directly related to the autocorrelation value, which is logical. Indeed, an autocorrelation of 0 means that after the product of the code and its delayed version, there are as many +1 as –1, which means that 10 chips over 20 are identical, and 10 chips are different. An autocorrelation of 4 means that there are twelve +1 and eight –1, which means 12 chips over 20 are identical, and 8 chips are different, and so on.

Table 4 also shows that from a delay to the one right after, there are 10 different chips. Therefore, if we want to compute $\mathbf{y}_0$, $\mathbf{y}_1$, $\mathbf{y}_2$, etc., 19 additions would be required to compute $\mathbf{y}_0$, and then 10 additions would be required for each $\mathbf{y}_k$, i.e. $19 + 10 \times 19 = 209$ additions in total, which is more than the previous methods. To reduce the number of additions, the difference between two delays computed consecutively should be 8 or 12, because with these delays there are only 8 different chips. The problem is that it will not be possible to go through all the delays because 8 and 20 are not relatively prime, e.g. starting from $\mathbf{y}_0$, the next results will be $\mathbf{y}_8$, $\mathbf{y}_{16}$, $\mathbf{y}_4$, $\mathbf{y}_{12}$, and then it will be again $\mathbf{y}_0$. Therefore, five delays can be computed in this way, but after another difference of

Table 4 : L5 secondary code recursive relations.

Delay	1	2	3	4	5	6	7	8	9
# chips different	10	10	10	10	10	12	10	8	10
Autocorrelation	0	0	0	0	0	–4	0	4	0

Delay	10	11	12	13	14	15	16	17	18	19
# chips different	12	10	8	10	12	10	10	10	10	10
Autocorrelation	–4	0	4	0	–4	0	0	0	0	0

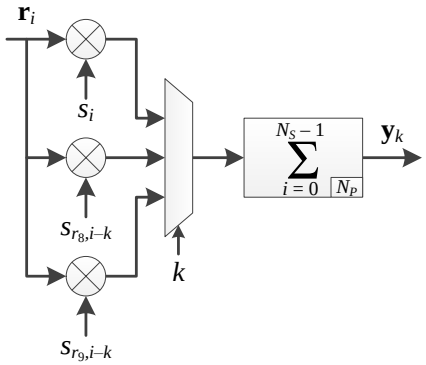

Figure 5 : Implementation of the method #3.

delay must be selected, for example 9, which require 10 additions. This way, after $\mathbf{y}_{12}$, $\mathbf{y}_1$ will be computed, and then five delays separated of 8 chips can be computed, and so on. Thus, $\mathbf{y}$ could be computed in the following order : $\mathbf{y}_0$, $\mathbf{y}_8$, $\mathbf{y}_{16}$, $\mathbf{y}_4$, $\mathbf{y}_{12}$, $\mathbf{y}_1$, $\mathbf{y}_9$, $\mathbf{y}_{17}$, $\mathbf{y}_5$, $\mathbf{y}_{13}$, $\mathbf{y}_2$, $\mathbf{y}_{10}$, $\mathbf{y}_{18}$, $\mathbf{y}_6$, $\mathbf{y}_{14}$, $\mathbf{y}_3$, $\mathbf{y}_{11}$, $\mathbf{y}_{19}$, $\mathbf{y}_7$, $\mathbf{y}_{15}$, which requires $19 + 4 \times 8 + (10 + 4 \times 8) \times 3 = 177$ additions. The number of additions is thus reduced by 53.4 % compared to the traditional implementation, which is slightly better than method #1, but lower than method #2. Therefore, the output can be computed as

$$\mathbf{y}_k = \begin{cases} \sum_{i=0}^{N_S-1} s_i\,\mathbf{r}_i\,, & \text{for } k = 0 \\ \mathbf{y}_{k-9} + \sum_{i=0}^{N_S-1} s_{r_9,i-k}\,\mathbf{r}_i\,, & \text{for } k = 1,2,3 \\ \mathbf{y}_{k-8} + \sum_{i=0}^{N_S-1} s_{r_8,i-k}\,\mathbf{r}_i\,, & \text{else.} \end{cases} \quad \textbf{(15)}$$

Matlab simulations show a reduction of the processing time of about 36.5 %, as shown in Figure 6. As for the hardware implementation, 178 accesses to $\mathbf{r}_i$ will be needed, giving a reduction of the processing time of about 55.5 %, which is slightly better than method #1, but lower than method #2. However, there is a great advantage compared to methods #1 and #2, no additional memory is required to store intermediate results, as shown in Figure 5. Therefore the gain in processing time comes at almost no cost (only the control signals will be slightly more complicated, but this will not have a significant impact on resources). Nevertheless, it would be possible to duplicate the implementation of Figure 5, to compute two outputs simultaneously. In this case, the resources would be similar to those of method #1 since there is an additional memory (the other elements being negligible), but now only $20 + 4 \times 8 + 10 + 4 \times 8 = 94$ accesses are needed, which means a reduction of the processing time of 76.5 % compared to the traditional implementation. Therefore, with the L5 signal, method #3 is clearly more interesting than the other methods, because it provides

better performance and more flexibility with the possibility of compromise between processing time and resources.

For the E1 secondary code, the autocorrelation is 1 or –3, therefore there are at most 13 chips identical for different delays, i.e. 12 chips different. Computing a new $\mathbf{y}_k$ will then requires 12 additions, against 10 with method #1, thus this method is not interesting for E1. Likewise for the E5 secondary codes, where 46 additions would be required to compute a new $\mathbf{y}_k$, against 45 with method #1.[1]

5. CONCLUSION

To have high sensitivity with modern GNSS signals, long coherent integration time is needed, and thus it is necessary to acquire the secondary code. This paper proposes three methods to do so while reducing the complexity, which are summarized in Table 5 .

The first method, adding ±1 to the secondary code, is simple to apply to any code and reduces the complexity by at least 50 % in theory, at the expense of the storage of an intermediate result. The second method, adding an efficient code to the code, is applicable to the L5 and E5 signals, and allows a greater reduction of the complexity, up to 63 %, at the expense of the storage of one or two intermediate results. Finally, the third method, using recursive computation, is applicable only to the L5 signal, and provides the same theoretical complexity as the first method, but it does not require the storage of any intermediate result. However, a storage could be added to reduce even more the processing time for hardware implementations.

The reductions of the processing time observed in Matlab simulations are a bit less than foreseen by the number of operations, but still significant. The reduction of the processing time for hardware implementation is easily evaluated and is close to the theoretical reductions.

The best choice between these methods will depend on the context, e.g., if the architecture should be process only one signal or several signals, which signal(s), or if an additional memory can be added or not. For example, for a hardware receiver, if only the L5 signal wants to be acquired and the resources want to be minimized, the third method is the best choice; if L5, E1 and E5 signals want to be acquired, it is possible to implement the three methods using one additional memory, since the structure of the implementations is similar.

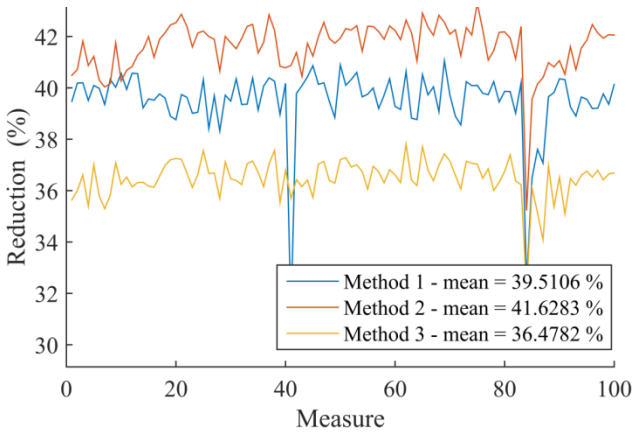

Figure 6 : Reduction of the processing time of the propose methods measured with Matlab simulations.

APPENDIX. Z-TRANSFORM AND DFT EXPRESSIONS OF TIERED CODE

The z-transform of the tiered code $c[n]$ is defined as

$$C(z) = \sum_{n=0}^{N-1} c[n]\, z^{-n}, \tag{16}$$

with $z \in \mathbb{C}$. Using $n = n_p + N_P n_s$ and $c[n] = p[n_p]s[n_s]$, Eq. (16) becomes

$$\begin{aligned} C(z) &= \sum_{n_p=0}^{N_P-1} \sum_{n_s=0}^{N_S-1} p[n_p]\, s[n_s]\, z^{-(n_p+N_P n_s)} \\ &= \sum_{n_p=0}^{N_P-1} p[n_p]\, z^{-n_p} \sum_{n_s=0}^{N_S-1} s[n_s]\, z^{-N_P n_s} \\ &= P(z)\, S(z^{N_P}). \end{aligned} \tag{17}$$

Table 5 : Summary of the performances of the proposed methods with the L5 signal.

	Method #1	Method #2	Method #3
Reduction of additions	52.9 %	63.1 %	53.4 %
Reduction of processing time in Matlab	39.5 %	41.6 %	36.5 %
Reduction of processing time in hardware	55 % with one memory	60 % with one memory 65 % with two memories	55.5 % without memory 76.5 % with one memory
Advantages	Very simple, Applicable to any code	Simple, Applicable to L5 and E5 codes	Additional memory not required
Drawbacks	Requires an additional memory	Requires one or two additional memories	Applicable to L5 only,[1] Control signals more complex

[1] Actually, this is not true. Recursion is interesting for E1 and E5 signals for hardware implementations, because the processing time can still be reduced at no cost, i.e. without additional memory; and using one additional memory the reduction is greater than the other methods. Using recursion with one additional memory, there will be 25 + (12 × 24) / 2 = 169 accesses for E1 (275 with method #1), and 100 + (46 × 99) / 2 ≈ 2400 accesses for E5 (4756 with method #1).In conclusion, recursion is the most interesting method for hardware implementations.(note added in April 2017).

The DFT of the tiered code $c[n]$ is defined as

$$C[k] = C\left(z = e^{\frac{j2\pi k}{N}}\right) = \sum_{n=0}^{N-1} c[n]\, e^{-\frac{j2\pi kn}{N}}, \qquad \textbf{(18)}$$

with $j^2 = -1$. Using Eq. (17), Eq. (18) becomes

$$\begin{aligned} C[k] &= \sum_{n_p=0}^{N_P-1} p[n_p]\, e^{-\frac{j2\pi kn_p}{N}} \sum_{n_s=0}^{N_S-1} s[n_s]\, e^{-\frac{j2\pi kN_Pn_s}{N}} \\ &= \sum_{n_p=0}^{N-1} p_Z[n_p]\, e^{-\frac{j2\pi kn_p}{N}} \sum_{n_s=0}^{N_S-1} s[n_s]\, e^{-\frac{j2\pi kn_s}{N_S}} \\ &= P_Z[k]\, S[k], \end{aligned} \qquad \textbf{(19)}$$

where p_Z is p padded with zeros to reach a length of N. This means that the DFT of the tiered can be computed as the product between the DFT of the zero-padded primary code and the DFT of the secondary code (this last one is shorter but DFTs are periodic, thus $S[k + N_S] = S[k]$).

REFERENCES

[1] F. Van Diggelen, "A-GPS: Assisted GPS, GNSS, and SBAS", Artech House, 2009.

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

[3] C.J. Hegarty, "GNSS signals — An overview", *IEEE International Frequency Control Symposium*, pp. 1-7, May 2012.

[4] 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.

[5] K. Borre, D. Akos, N. Bertelsen, P. Rinder, S. Jensen, "A software-defined GPS and Galileo receiver: A single-frequency approach", Birkhauser Boston, 2007.

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

[7] M. Foucras, "Performance analysis of the modernized GNSS signal acquisition", Ph.D. thesis, INP Toulouse, France, 2015.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] K. Sun, L. Lo Presti, "Bit sign transition cancellation method for GNSS signal acquisition", *Journal of Navigation*, vol. 65, pp. 73-97, 2012.

[14] 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.

[15] J. Leclère, C. Botteron, P.-A. Farine, "Acquisition of modern GNSS signals using a modified parallel code-phase search architecture," *Signal Processing*, vol. 95, pp. 177-191, 2014.

[16] 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.

[17] M. Foucras, O. Julien, C. Macabiau, B. Ekambi, F. Bacard, "Probability of detection for GNSS signals with sign transitions", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 3, pp. 1296-1308, 2016.

[18] J. Leclère, C. Botteron, "Expressing discrete convolutions and correlations using matrices and polynomials: a unified presentation", *IEEE Signal Processing Magazine*, submission in October 2016.

[19] J. Leclère, C. Botteron, P.-A. Farine, "High sensitivity acquisition of GNSS signals with secondary code on FPGAs", *IEEE Aerospace & Electronics Systems Magazine*, submitted July 2016.

[20] S.W. Smith. "Digital signal processing: a practical guide for engineers and scientists", Newnes, 2002.

[21] S. Winograd, "Arithmetic complexity of computations", Society for Industrial and Applied Mathematics, 1980.

[22] H.J. Nussbaumer, "Fast Fourier transform and convolution algorithms", Springer Berlin Heidelberg, 2nd edition, 1982.

[23] C.S. Burrus, T.W. Parks, "DFT/FFT and convolution algorithms and implementation", Wiley, 1985.

[24] R.E. Blahut. "Fast algorithms for signal processing", Cambridge University Press, 2010.