High sensitivity acquisition of GNSS signals with secondary code on FPGAs

Jérôme Leclère[1], Cyril Botteron[2], *Senior Member, IEEE,* and Pierre-André Farine[2], *Member, IEEE*

***Abstract*—The presence of a secondary code in modern global navigation satellite system signals complicates the acquisition of these signals, because there is a potential sign transition between each period of the primary code. Some previous works proposed to use the parallel code search by performing the correlation over the primary code several times and then combining the results according to the secondary code chips. In this article, we will focus on this method and compare different hardware implementations, to determine if it is better to do the combinations before or after the correlations, and to compare serial and parallel architectures. In a second part, we will show a simple method that manipulates the local secondary code to rearrange the equations, which approximately halves the theoretical number of operations related to the secondary code correlation and the processing time for hardware implementations, without any impact on the sensitivity.**

***Index Terms*—Acquisition, FFT, FPGA, GNSS, Secondary code.**

I. Introduction

THE modern global navigation satellite systems (GNSS) signals, such as the GPS L5 and L1C, and Galileo E5 and E1, have brought several innovations : the introduction of a pilot channel that does not contain any data to allow very long coherent integrations; the introduction of a secondary code to offer better cross-correlations, to facilitate the synchronization with the data, and to help interference mitigation; the introduction of new modulations to reduce the impact of multipath; and the use of higher chipping rates to have better accuracy and interference mitigation.

Although having a secondary code brings some advantages, it also presents some drawbacks. Indeed, with the modern GNSS signals, there is now a potential sign transition (i.e. a carrier phase shift of $180°$) between each period of the primary code, unlike the GPS L1 C/A signal that has a potential sign transition each 20 code periods only. These sign transitions are one of the limitations of the coherent integration time, and thus of the receiver sensitivity [1], [2], [3], [4]. Therefore, to use a long coherent integration time and get high sensitivity, the delay of the secondary code must be estimated.

There have been several proposals to address this problem. In [5], [6], it has been proposed to synchronize with the primary code first, and then synchronize with the secondary code. However, this implies to be able to detect the signals using only one period of the primary code, which is not the case in the high sensitivity context. In [7], [8], it has been proposed to extend the coherent integration time by estimating the possible combinations of several secondary code chips, and using this to determine the secondary code delay [9], but these methods are still not adapted to the high sensitivity context. To get high sensitivity, the coherent integration time should be at least one period of the secondary code, or a multiple of it. In [10], it has been proposed to determine the primary code delay with a serial search and the secondary code delay with a fast Fourier transform (FFT) based correlation, however the serial search is too time-consuming for a realistic implementation. In [11], the authors proposed to perform an FFT-based correlation over one period of the secondary code with the L5 signal, nevertheless this requires very large FFTs (length greater than 2^{18}), which are not compatible with a hardware implementation. Finally, [12] proposed to perform FFT-based correlations over one period of the primary code (doubling the length to manage the sign transition), and to combine the results according to the secondary code chips.

In this article, we will focus on this last method. More specifically, we will compare different hardware implementations of this method. Indeed, the combinations can be performed before or after the correlations with the local primary code; they can be computed sequentially or in parallel; and the output can be computed in different orders (checking all the primary code delays for one secondary code delay, or checking all the secondary code delays for one primary code delay). The objective is therefore to identify the most efficient implementations. Note that these different implementations are not approximations, they all provide the same output and thus the same performance in terms of sensitivity. We will also present a method that approximately halves the number of operations related to the secondary code correlation, still without impacting the sensitivity, and see how it can reduce the processing time with the hardware implementations.

The article is organized as follows: Section II briefly introduces the GNSS signals, the parallel code search acquisition method, and the difficulties to perform the correlation over one period of the secondary code. Then, Section III recalls how to compute the correlation over the primary code and perform the combinations, and compares the different hardware implementations. Finally, Section IV presents the method to reduce the number of operations, and Section V concludes this paper.

II. Acquisition of GNSS signals

A. Signal definition

The signal received by a GNSS receiver is the combination of several GNSS signals coming from U different satellites,

[1] The author is with the Laboratory of Space technologies, Embedded systems, Navigation and Avionic (LASSENA) of the École de Technologie Supérieure (ÉTS), Canada. [2] The authors are with the Electronics and Signal Processing Laboratory (ESPLAB) of the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

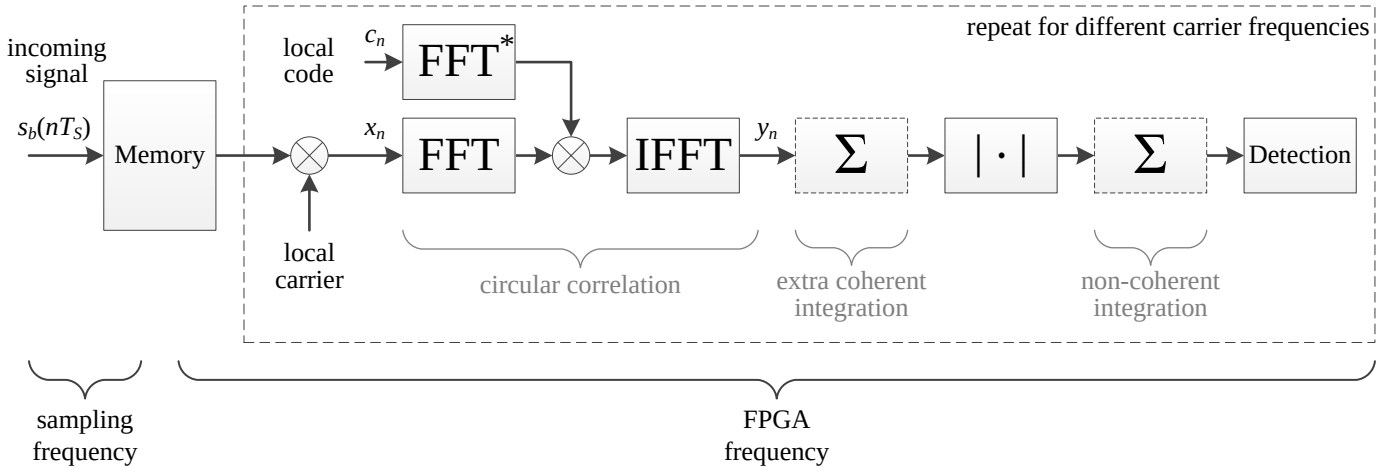

Fig. 1. Basic diagram of the parallel code search implemented on an FPGA, using a buffer for faster processing.

plus a noise term. Thus, after the front-end, the discrete baseband signal can be written as

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

where $s_b^u(nT_S)$ is the discrete baseband signal from satellite u, n is the discrete time index, T_S is the sampling period equal to $\frac{1}{f_S}$ with f_S being the sampling frequency, and $\eta_b(nT_S)$ is the noise component [13, Chap. 1].

Considering a real sampling front-end, the discrete baseband signal from satellite u having a data and a pilot channel can be expressed as

$$\begin{aligned} s_b^u(nT_S) = &\sqrt{2P_{b,d}^u} d^u\big(nT_S - \tau^u\big) c_d^u\big(nT_S - \tau^u\big) \\ &\cos\big(2\pi f_b^u nT_S + \varphi_{b,d}^u\big) \\ &+ \sqrt{2P_{b,p}^u} c_p^u\big(nT_S - \tau^u\big) \sin\big(2\pi f_b^u nT_S + \varphi_{b,p}^u\big), \end{aligned} \tag{2}$$

where $P_{b,d}^u$ and $P_{b,p}^u$ are the powers of the data and pilot channels, c_d^u and c_p^u are the pseudo random codes of the data and pilot channels, d^u is the data sequence, τ^u is an unknown delay, f_b^u is the baseband frequency that includes the intermediate frequency, a global offset caused by the local oscillator and the offset caused by the Doppler effect, and $\varphi_{b,d}^u$ and $\varphi_{b,p}^u$ are the carrier phases of the data and pilot channel [13], [14]. Note that this model is simplified, since it does not take into account the Doppler effect on the code, or the local oscillator effect on the sampling frequency for example (see [13, Chap. 1]), but it is enough for our problem.

The pseudo random codes c_d^u and c_p^u are composed of a primary code and of a secondary code (and potentially of a sub-carrier, not considered here but without impact on our discussion), and are also called tiered codes. In these tiered codes, the primary code is repeated several times and each period is multiplied by a chip of the secondary code. Since the primary and secondary codes are binary codes taking $+1$ or -1 as value, the tiered code is also binary code taking $+1$ or -1 as value. Using vector notation, denoting $\mathbf{p}$ the primary code of length N_P, and $\mathbf{s}$ the secondary code of length N_S, they can be defined as

$$\mathbf{p} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N_P-1} \end{bmatrix}, \quad \mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{N_S-1} \end{bmatrix}, \tag{3}$$

where the subscript represents the sample for $\mathbf{p}$ and the chip for $\mathbf{s}$. The tiered code, denoted $\mathbf{c}$, has thus a length $N = N_S N_P$ and is defined as

$$\mathbf{c} = \mathbf{s} \otimes \mathbf{p} = \begin{bmatrix} s_0\mathbf{p} \\ s_1\mathbf{p} \\ \vdots \\ s_{N_S-1}\mathbf{p} \end{bmatrix}, \tag{4}$$

where $\otimes$ denotes the Kronecker product. The length N_P of the primary code depends on the signal and on the sampling frequency. For example, the L5, E5a and E5b pilot signals are binary phase shift keying (BPSK) signals with a chipping rate of 10.23 MHz, therefore the usual minimum sampling frequency considered for these signals is 20.46 MHz (twice the chipping rate, but this exact frequency is never used because of position accuracy problems [15]). Since the length of the primary code is 1 ms, the minimum value of N_P is 20 460. The length N_S of the secondary code is not related to the receiver and depends only on the signal. For example, the length of the secondary codes on the data and pilot channels is respectively 10 and 20 chips for the L5 signal, 20 and 100 chips for the E5a signal, and 4 and 100 chips for the E5b signal. Therefore, the minimum value of N is 409 200 for the L5 pilot signal, and 2 046 000 for the E5a and E5b pilot signals.

B. Parallel code search acquisition

The aim of the acquisition is to detect the visible GNSS satellites, and to estimate their baseband frequency f_b^u and code delay τ^u, by synchronizing local replicas with the incoming signal. The acquisition is thus a two-dimensional problem, for each satellite. There are different methods to perform the acquisition, such as the serial search, which tests the different

combinations for the carrier frequency and code delay one by one [16]; the parallel frequency search, which tests one code delay and several or all the carrier frequencies in parallel using an FFT [17], [18], [19]; the parallel code search, which tests one carrier frequency and all the code delays in parallel using an FFT-based correlation [20], [16], [13]; or there are also methods that parallelize the search in the two dimensions [21], [22], [23]. For a high sensitivity hardware receiver, the parallel code search seems the most suitable method because of its high level of parallelization, its moderate memory requirements, and because it can compensate the code Doppler whereas the parallel frequency search and its derivates cannot [13], [24].

The basic diagram of the parallel code search implemented on an FPGA is shown in Fig. 1. In this figure, the incoming signal is stored in a memory at the sampling frequency for a faster processing during the acquisition. For different frequencies of the local carrier replica, the circular correlation between the incoming code and the local code is computed using FFTs. Then additional coherent or non-coherent integration can be performed. This process is performed at the FPGA frequency, which is usually much higher than the sampling frequency, allowing a speeding up of the acquisition [24].

For the following, we will concentrate only on the processing between the carrier removal and the extra coherent integration, i.e. the circular correlation computed using FFTs.

C. Direct correlation over the secondary code period

The circular correlation can be performed over the entire tiered code to synchronize with both primary and secondary codes simultaneously, as proposed in [11]. Using matrix notation, the circular correlation can be written as

$$\mathbf{y} = \mathbf{C}\mathbf{x} = \mathbf{X}\mathbf{c}, \tag{5}$$

where $\mathbf{C}$ is an $N \times N$ right circulant matrix with $\mathbf{c}^{\mathrm{T}}$ as first row, $\mathbf{x}$ is the signal after the carrier removal, and $\mathbf{X}$ is an $N \times N$ left circulant matrix with $\mathbf{x}$ as first column [25]. Since a circulant matrix can be diagonalized by the discrete Fourier transform matrix $\mathbf{F}$, we can write

$$\mathbf{y} = \mathbf{F}^{-1}\big((\mathbf{F}\mathbf{c})^* \circ (\mathbf{F}\mathbf{x})\big), \tag{6}$$

where * denotes the conjugate operator and $\circ$ denotes the Hadamard product (element by element product) [25]. Therefore, this circular correlation can be implemented using FFTs as shown in Fig. 2, where the length of the FFTs is N. The corresponding timing diagram is shown in Fig. 14 (all the timing diagrams are provided in appendix to not overload the core of the article), assuming that several FFTs can be computed consecutively without pause (this corresponds to the streaming implementation of some FFTs [26]), and that the FFT has a latency of L_N clock cycles (i.e. there are L_N clock cycles between the last sample of the input sequence and the first sample of the first output sequence).

For an FPGA implementation, the FFT cores available require an FFT length that is a power of two [26], [27], [28], [29]. As mentioned previously, the minimum value of N is 409 200 for the L5 signal, thus the smallest power of two possible is $2^{19} = 524\,288$. To have this FFT length, the sampling frequency must be 26.2144 MHz (524 288 / 20 ms). Otherwise, if another sampling frequency is considered, zero-padding must be used, and the equivalent of two code periods are needed (to keep the periodicity of the code and avoid losses [30], [31]), and in this case the FFT length would be 1 048 576.

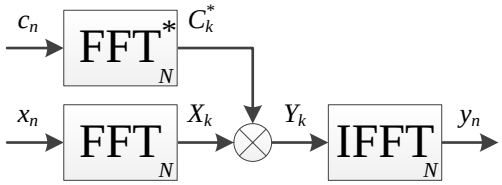

Fig. 2. Implementation of the direct correlation over the secondary code period (Eqs. (5) and (6)). See details in Section II-C, and the timing diagram in Fig. 14.

In any cases, it is not possible to implement such FFT directly since the required length is too large. Indeed, the maximum length currently available with the Altera FFT core is 262 144 with the variable streaming data flow (which consumes a tremendous amount of resources) and 65 536 with the streaming and burst data flows [26]; the maximum length is 65 536 with the Xilinx FFT core [27]; 16 384 with the Lattice FFT core [28]; and 8192 with the Microsemi FFT core [29]. Nevertheless, the processing time of the theoretical implementation of the direct correlation is given in Table I, without and with zero-padding. In the next section, we will consider the computation of the circular correlation by combining the results of smaller circular correlations, which is more practical for hardware implementations. The processing time and memory usage of all the implementations are given in Table I.

III. Correlation over the primary code period and combinations

Instead of computing the circular correlation over one entire period of the secondary code, it is possible to perform a circular correlation over one period of the primary code, repeat this for multiple consecutive periods, and then combine the results according to the chips of the secondary code, as proposed in [12]. Therefore the output $\mathbf{y}$ can be computed by portions equivalent to the period of the primary code. For example, considering that the secondary code has four chips (example that we will use along this paper for the illustrations), the four portions of the output can be obtained as

$$\mathbf{y} = \begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \mathbf{y}_2\\ \mathbf{y}_3\end{bmatrix} = \begin{bmatrix} s_0\mathbf{P}_T\mathbf{x}_0 + s_1\mathbf{P}_T\mathbf{x}_1 + s_2\mathbf{P}_T\mathbf{x}_2 + s_3\mathbf{P}_T\mathbf{x}_3\\ s_3\mathbf{P}_T\mathbf{x}_0 + s_0\mathbf{P}_T\mathbf{x}_1 + s_1\mathbf{P}_T\mathbf{x}_2 + s_2\mathbf{P}_T\mathbf{x}_3\\ s_2\mathbf{P}_T\mathbf{x}_0 + s_3\mathbf{P}_T\mathbf{x}_1 + s_0\mathbf{P}_T\mathbf{x}_2 + s_1\mathbf{P}_T\mathbf{x}_3\\ s_1\mathbf{P}_T\mathbf{x}_0 + s_2\mathbf{P}_T\mathbf{x}_1 + s_3\mathbf{P}_T\mathbf{x}_2 + s_0\mathbf{P}_T\mathbf{x}_3 \end{bmatrix}, \tag{7}$$

where the $\mathbf{y}_i$ are the different portions of the output containing N_P samples, $\mathbf{P}_T$ is a Toeplitz matrix of size $N_P \times 2N_P$ where the first row is the primary code $\mathbf{p}^{\mathrm{T}}$ padded with N_P zeros, and the $\mathbf{x}_i$ are built from two consecutive periods of the incoming code, i.e. $\mathbf{x}_i = \begin{bmatrix} x_{i\times N_P} & x_{i\times N_P+1} & \cdots & x_{i\times N_P+2N_P-1}\end{bmatrix}^{\mathrm{T}}$, they contain thus $2N_P$ samples (see [13, Chap. 6] for more details about how to obtain this equation). Note that (7) is not

an approximation of (5), the output $\mathbf{y}$ is exactly the same in both cases. Only the way to compute $\mathbf{y}$ is different.

The Toeplitz matrix $\mathbf{P}_T$ can be embedded into a circulant matrix of size $2N_P \times 2N_P$ by adding N_P rows [25], therefore the product between $\mathbf{P}_T$ and a vector of $2N_P$ points can be computed as a circular correlation using three FFTs of length $2N_P$, where the second half of the output is discarded. If the length of the FFTs has a constraint (such as to be a power of two), zeros can be added to the local and incoming sequences to achieve the desired length. Since we focus on FPGA implementations, we consider this constraint, and thus for the acquisition of the L5, E5a and E5b signals, the length of the FFT will be $N_{FFT} = 2N_P + N_Z = 65\,536$ (since the minimum for $2N_P$ is 40 920). This value for the FFT length can be used for sampling frequencies up to 32.768 MHz. Note that there are methods to optimize this double length circular correlation for FPGA implementations [31], [32], but we do not consider them in the following discussions since this circular correlation is present in all the implementations discussed.

Coming back to (7), the multiplication by the secondary code chips can be done at different stages. If the different combinations according to the secondary code delay are performed before the FFT-based correlation, (7) becomes

$$\begin{aligned}\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \mathbf{y}_2\\ \mathbf{y}_3\end{bmatrix} &= \begin{bmatrix}\mathbf{P}_T & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{P}_T & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{P}_T & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}_T\end{bmatrix} \begin{bmatrix}s_0 & s_1 & s_2 & s_3\\ s_3 & s_0 & s_1 & s_2\\ s_2 & s_3 & s_0 & s_1\\ s_1 & s_2 & s_3 & s_0\end{bmatrix} \begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\\ \mathbf{x}_2\\ \mathbf{x}_3\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{P}_T & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{P}_T & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{P}_T & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}_T\end{bmatrix} \begin{bmatrix}\mathbf{a}_0\\ \mathbf{a}_1\\ \mathbf{a}_2\\ \mathbf{a}_3\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{P}_T\mathbf{a}_0\\ \mathbf{P}_T\mathbf{a}_1\\ \mathbf{P}_T\mathbf{a}_2\\ \mathbf{P}_T\mathbf{a}_3\end{bmatrix},\end{aligned} \tag{8}$$

where $\mathbf{a}_j = \sum_{i=0}^{N_S-1} s_{((i-j))}\mathbf{x}_i$, the double parenthesis meaning a modulo-N_S operation. Since the secondary code is removed before the FFTs, we will talk about pre-FFT secondary code removal. If the different combinations are performed after the FFT-based correlation, (7) becomes

$$\begin{aligned}\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \mathbf{y}_2\\ \mathbf{y}_3\end{bmatrix} &= \begin{bmatrix}s_0 & s_1 & s_2 & s_3\\ s_3 & s_0 & s_1 & s_2\\ s_2 & s_3 & s_0 & s_1\\ s_1 & s_2 & s_3 & s_0\end{bmatrix} \begin{bmatrix}\mathbf{P}_T\mathbf{x}_0\\ \mathbf{P}_T\mathbf{x}_1\\ \mathbf{P}_T\mathbf{x}_2\\ \mathbf{P}_T\mathbf{x}_3\end{bmatrix}\\ &= \begin{bmatrix}s_0 & s_1 & s_2 & s_3\\ s_3 & s_0 & s_1 & s_2\\ s_2 & s_3 & s_0 & s_1\\ s_1 & s_2 & s_3 & s_0\end{bmatrix} \begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \mathbf{r}_2\\ \mathbf{r}_3\end{bmatrix},\end{aligned} \tag{9}$$

where $\mathbf{r}_i = \mathbf{P}_T\mathbf{x}_i$. Since the secondary code is removed after the FFTs, we will talk about post-FFT secondary code removal. The notation $\mathbf{a}_i$ and $\mathbf{r}_i$ are used to facilitate the link between the equations and the figures. Note that $\mathbf{r}_i$ has a clear meaning, since we can write (assuming the Doppler is correctly removed)

$$\mathbf{r}_i = s_{i-\Delta}\,\mathbf{r}_p + \boldsymbol{\eta}_i, \tag{10}$$

where Δ is the unknown delay of the incoming secondary code, $\mathbf{r}_p$ is the autocorrelation of the primary code, and $\boldsymbol{\eta}_i$ is the noise.

In both equations (8) and (9), there are N_S FFT-based correlations over at least $2N_P$ points. For the combinations, (8) uses vectors of $2N_P$ points, whereas (9) uses vectors of N_P points. Thus, (9) requires slightly less operations than (8). One can check that these two equations require more operations than the direct correlation of (5).

In the next subsections, we will study the FPGA implementation of both equations, testing the secondary code delays sequentially or in parallel, and using or not memory to save temporary results. For the evaluation of the processing time, we will consider a streaming data flow, i.e. an FFT that can process the data in a continuous way.

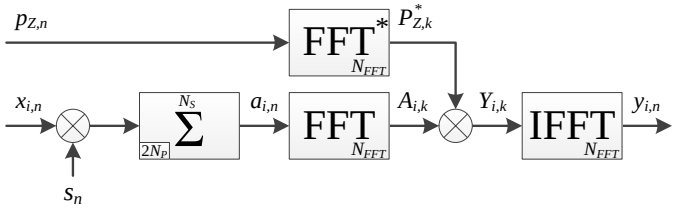

Fig. 3. Implementation of the pre-FFT secondary code removal (Eq. (8)) computing each combination of the input sequentially. See details in Section III-A, and the timing diagram in Fig. 15.

A. Implementation of the pre-FFT secondary code removal in a sequential way

In (8), there are N_S correlations between the local primary code $\mathbf{p}$ and the combinations of the different periods of the incoming signal ($\mathbf{a}_0$, $\mathbf{a}_1$, ...). The corresponding implementation computing each combination sequentially is shown in Fig. 3. The accumulator used before the FFT is implemented with an adder and a memory having $2N_P$ addresses, to accumulate over N_S samples (one sample of each period), as shown in Figs. 12 and 13. The processing starts by accessing all the portions of the input ($\mathbf{x}_0$, $\mathbf{x}_1$, ...), and when the last one is accessed, the first combination $\mathbf{a}_0$ is available and its correlation with the local code $\mathbf{p}$ is computed to obtain $\mathbf{y}_0$. Then, $\mathbf{x}_0$, $\mathbf{x}_1$, ... can be accessed again immediately to compute the second combination, and so on and so forth, until the N_S combinations have been tested. The processing time is approximately $2N_S$ times higher than the one of the direct correlation implementation.

With this implementation, the memory needed is twice $2N_P\,(B_0 + \lceil \log_2 N_S \rceil)$ bits for the accumulation because the signal is complex, where B_0 denotes the number of bits used to quantize $\mathbf{x}_i$.

B. Implementation of the pre-FFT secondary code removal in a parallel way

It is also possible to compute the different combinations ($\mathbf{a}_0$, $\mathbf{a}_1$, ...) in parallel using N_S accumulators, as shown in Fig. 4.

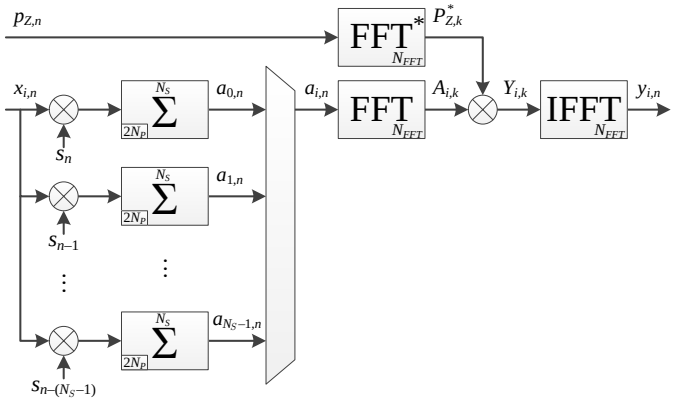

Fig. 4. Implementation of the pre-FFT secondary code removal (Eq. (8)) computing each combination of the input in parallel. See details in Section III-B, and the timing diagram in Fig. 16.

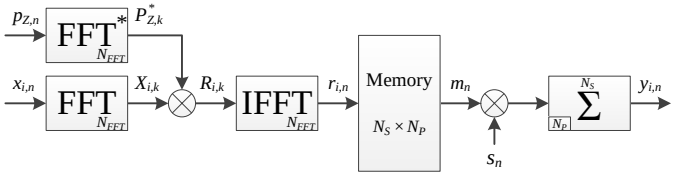

Fig. 5. Implementation of the post-FFT secondary code removal (Eq. (9)) using a memory to store correlation portions and computing each combination of the output sequentially. See details in Section III-C, and the timing diagram in Fig. 18.

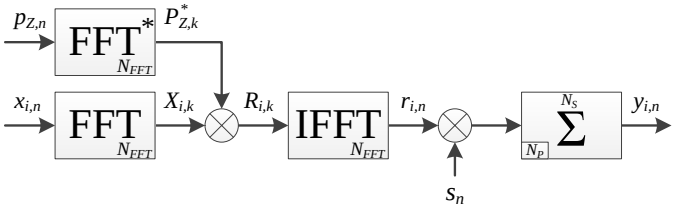

Fig. 6. Implementation of the post-FFT secondary code removal (Eq. (9)) computing each combination of the output sequentially. See details in Section III-D, and the timing diagram in Fig. 19.

In this case, the processing also starts by accessing $\mathbf{x}_0$, $\mathbf{x}_1$, ..., and when the last one is accessed, $\mathbf{a}_0$, $\mathbf{a}_1$, ... are available in the accumulators memory. Then, each $\mathbf{a}_i$ is read successively and the correlation with the local code $\mathbf{p}$ is computed to obtain $\mathbf{y}_i$. Then, for the next data stream, the portions of the input can be accessed again only when the last combination is read, which implies that the processing time is divided by a factor lower than N_S compared to the previous sequential implementation.

With this implementation, the memory requirements are higher since $2N_P N_S \left(B_0 + \lceil \log_2 N_S \rceil\right)$ bits need to be stored for the accumulation.

These two pre-FFT implementations are the extreme cases, where either only one combination is computed at a time, or all the combinations are computed simultaneously. However, it is also possible to test only several combinations, using one accumulator per combination. For example, the timing diagram considering two accumulators is given in Fig. 17.

C. Implementation of the post-FFT secondary code removal with a memory

Looking at (9), it can be seen that the correlation between the local primary code $\mathbf{p}$ and each portion of the incoming code ($\mathbf{x}_0$, $\mathbf{x}_1$, ...) needs to be performed only once. Only the combinations of the different portions according to the secondary code delays differs. However, this requires to store the correlation portions ($\mathbf{r}_0$, $\mathbf{r}_1$, ...). The corresponding implementation using a memory to store $\mathbf{r}_0$, $\mathbf{r}_1$, ..., and computing $\mathbf{y}_0$, $\mathbf{y}_1$, ... sequentially is shown in Fig. 5.

The processing starts by accessing $\mathbf{x}_0$, $\mathbf{x}_1$, ..., computing their correlation with the local code $\mathbf{p}$, and storing the results into the memory. Then, the memory is read and a combination is tested, then the memory is read again and another combination is tested, and so on and so forth. The process is then repeated for the next data stream, as soon as it is possible to write again into the memory without overwriting data not yet read. With this implementation, the combinations are performed over vectors of N_P instead of $2N_P$ for the pre-FFT implementations, which implies that the processing time is approximately halved compared to the pre-FFT sequential implementation.

With this implementation, the memory needed is twice $N_P N_S \times B_1$ bits to store the FFT outputs and twice $N_P \left(B_1 + \lceil \log_2 N_S \rceil\right)$ bits for the accumulation, where B_1 denotes the number of bits used to quantize the outputs of the IFFT ($\mathbf{r}_i$).

D. Implementation of the post-FFT secondary code removal in a sequential way

It is also possible to implement (9) without storing $\mathbf{r}_0$, $\mathbf{r}_1$, ..., but in this case they must be recomputed several times. The corresponding implementation computing $\mathbf{y}_0$, $\mathbf{y}_1$, ... sequentially is shown in Fig. 6.

The processing starts by accessing $\mathbf{x}_0$, $\mathbf{x}_1$, ..., computing their correlation with the local code $\mathbf{p}$, and combining the results according to the secondary code chips. The process is then repeated to test the next combinations. Then, the process is repeated for the next data streams. With this implementation, since the zero-padding is present at the input of the FFTs and for the combinations, the processing time is higher than with the pre-FFT sequential implementation and with the post-FFT implementation with a memory.

E. Implementation of the post-FFT secondary code removal in a parallel way

As previously, it is also possible to compute each portion of the output in parallel using N_S accumulators, as shown in Fig. 7. The processing is similar to the previous post-FFT implementation, except that the FFTs are computed only once, since each accumulator accumulates when a new correlation portion is available, and that there are N_S output available simultaneously (which will require a slightly different detection process after that). Contrary to the pre-FFT parallel implementation, there is no need to stop the stream between different data streams, therefore the processing time is lower. Note also that the processing time is divided by approximately N_S compared to the post-FFT sequential implementation.

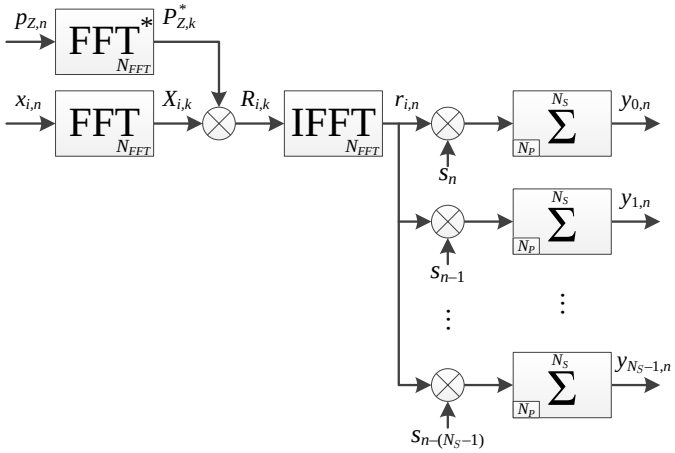

Fig. 7. Implementation of the post-FFT secondary code removal (Eq. (9)) computing each combination of the output in parallel. See details in Section III-E, and the timing diagram in Fig. 20.

F. Implementation of the post-FFT secondary code removal using circular correlation

In the previous implementations, the output is computed by consecutive portions corresponding to one primary code period. However it is also possible to compute the output in a different order. Indeed, the lth samples of the outputs $\mathbf{y}_0$, $\mathbf{y}_1$, ..., can be obtained from the circular correlation between the secondary code and the lth samples of the correlation portions ($\mathbf{r}_0$, $\mathbf{r}_1$, ...). Starting from (9), we can write

$$\begin{bmatrix} y_{0,l} \\ y_{1,l} \\ y_{2,l} \\ y_{3,l} \end{bmatrix} = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 \\ s_3 & s_0 & s_1 & s_2 \\ s_2 & s_3 & s_0 & s_1 \\ s_1 & s_2 & s_3 & s_0 \end{bmatrix} \begin{bmatrix} r_{0,l} \\ r_{1,l} \\ r_{2,l} \\ r_{3,l} \end{bmatrix}. \tag{11}$$

This circular correlation can be computed traditionally in the time domain, or using FFTs. However, this means that we need to have access to the different correlations portions at the same time, therefore, they should be stored as in Section III-C.

1) Implementation of the secondary code circular correlation in a sequential way: If the different combinations in (11) are computed in a sequential way, the accumulation can be done with a simple adder, without using a memory. The corresponding implementation is shown in Fig. 8.

The processing until the storage of the correlation portions is similar to the post-FFT implementation with a memory. After, what is different is the reading order of the memory, because now we read the first sample of each correlation portion ($r_{0,0}, r_{1,0}, ...$), multiply them with the secondary code and accumulate the result. These samples are then accessed again to test another delay of the secondary code, and so on and so forth. Thus they will be accessed N_S times. Then, we read the second sample of each portion ($r_{0,1}, r_{1,1}, ...$) and the same process is performed, and this is repeated N_P times for the N_P delays of the primary code.

Because of the different writing and reading order of the memory, there is an additional latency introduced compared to the post-FFT implementation with a memory (this can be clearly seen comparing Figs. 18 and 21), and therefore the processing time is slightly longer.

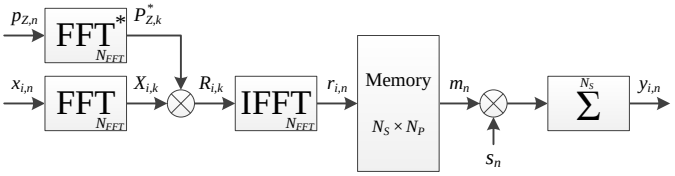

Fig. 8. Implementation of the post-FFT secondary code removal using circular correlation (Eq. (11)) computing each sample of the output sequentially (the writing and reading orders of the memory are different). See details in Section III-F1, and the timing diagram in Fig. 21.

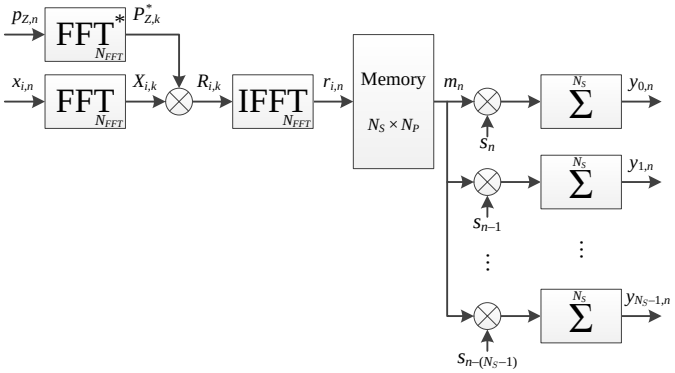

Fig. 9. Implementation of the post-FFT secondary code removal using circular correlation (Eq. (11)) computing each sample of the output in parallel (the writing and reading orders of the memory are different). See details in Section III-F2, and the timing diagram in Fig. 22.

With this implementation, the memory needed is twice $N_P N_S \times B_1$ bits to store the FFT outputs.

2) Implementation of the secondary code circular correlation in a parallel way: It is also possible to compute the N_S samples of the output in (11) in parallel using N_S accumulators, as shown in Fig. 9.

The processing until the storage of the correlation portions is similar to the previous implementation. The only difference is that we need to read only once the N_S samples $r_{0,l}, r_{1,l}, ...,$ to test the N_S combinations. Therefore, compared to the previous implementation, the processing time is reduced a lot (up to $N_S/3$) in exchange of only N_S logic accumulators. However, compared to the post-FFT parallel implementation, the processing time is slightly higher because of the different order of writing and reading in the memory that introduces a latency (this can be seen comparing Figs. 20 and 22).

3) Implementation of the secondary code circular correlation using FFTs: As indicated previously, since (11) corresponds to a circular correlation, the operation can be performed using FFTs. The corresponding implementation is shown in Fig. 10, where $N_{FFT,S}$ denotes the length of these small FFTs. Following our constraints, these FFTs need sequences that have a length that is a power of two. None of the secondary code currently available has such a length (except on the data channel of the E5b signal). Therefore, zero-padding must be used, and the length of the sequences must at least double (to keep the periodicity and avoid losses). For example, with the GPS L5 pilot secondary code that has

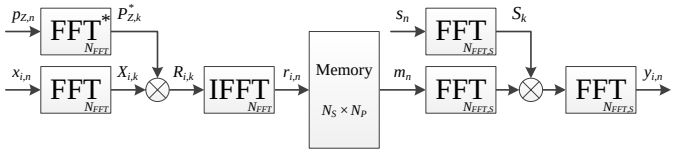

Fig. 10. Implementation of the post-FFT secondary code removal using circular correlation (Eq. (11)) computing each sample of the output using an FFT (the writing and reading orders of the memory are different). See details in Section III-F3, and the timing diagram in Fig. 23.

20 bits, the FFTs length will be 64 bits.

The process is similar to the previous implementation, except that more samples are needed to compute the circular correlation, and therefore the processing time is longer. Moreover, the resources required by an FFT of 64 points in terms of logic, memory and multipliers are not negligible, therefore such FFT will likely require more resources than the implementation of N_S accumulators (except maybe if $N_S = 100$, as with the E5a and E5b signals). Consequently, the use of the FFT for the circular correlation over the secondary code is not recommended.

G. Summary

Table I provides a summary of the memory needed and of the processing time for each considered implementation. Let's first have a look on the sequential implementations. Comparing the pre-FFT and post-FFT sequential implementations (Figs. 3 and 6), the second one requires a higher processing time due to the zero-padding (this extra time can be significant if N_Z is large), and its required memory is multiplied by $\frac{B_1+\lceil \log_2 N_S \rceil}{2(B_0+\lceil \log_2 N_S \rceil)}$. Usually, B_0 is rather small (since the incoming signal is typically quantized with 2 bits and the local carrier replica as well [33]), and B_1 is not small because the FFT requires a certain number of bits to provide accurate results (typically 16 bits, from experience). Thus, the memory requirements for both implementations can be relatively close. Therefore, the pre-FFT sequential implementation seems more interesting than the post-FFT sequential implementation.

For the post-FFT sequential implementation using a memory (Fig. 5), its processing time is roughly half the one of the post-FFT sequential implementation (Fig. 6), whereas the memory is multiplied by a factor close to N_S. Note however that the FFTs require a significant amount of memory, and that the incoming signal is also stored (see Fig. 1), therefore the total amount of memory needed for the acquisition is multiplied by a factor less than N_S. For the post-FFT implementation using a memory with a sequential secondary code circular correlation (Fig. 8), there is a slight increase in the processing time and a slight decrease in the memory requirements. Thus, the most suitable of these three post-FFT sequential implementations will depend on the context and design constraints.

Let's now compare the parallel implementations. Comparing the pre-FFT and post-FFT parallel implementations (Figs. 4 and 7), the second one has a lower processing time (by a factor at most two), whereas the memory is multiplied again by a factor $\frac{B_1+\lceil \log_2 N_S \rceil}{2(B_0+\lceil \log_2 N_S \rceil)}$. Therefore, there is probably an advantage for the post-FFT implementation, but the context and the design should be taken into account to make a precise evaluation.

For the post-FFT implementation using a memory with a parallel secondary code circular correlation (Fig. 9), its processing time is higher than the one of the post-FFT parallel implementation (Fig. 7) by a factor less than $\frac{3}{2}$, whereas its memory is multiplied by a factor $\frac{B_1}{B_1+\lceil \log_2 N_S \rceil}$, which is smaller than one. Therefore, it is again difficult to decide between these two implementations without more information about the context and the design.

For the post-FFT implementation using a memory with an FFT-based secondary code circular correlation (Fig. 10), the processing time is longer than the one of the post-FFT implementation using a memory with a parallel secondary code circular correlation (Fig. 9) by a factor of at least $\frac{4}{3}$, and the memory requirement is slightly higher due to the small FFTs. Therefore, this implementation is less efficient and not interesting.

To have a more concrete evaluation, let's consider two examples, one corresponding to a "low-cost" receiver where the incoming signal is quantized with few bits and sampled with a low frequency, and one corresponding to a "high-end" receiver using more bits for the quantization and a higher sampling frequency. The parameters selected considering the GPS L5 pilot signal are shown in Table II, and the results are shown in Tables III and IV.

For the evaluation of the memory required by the FFTs, we have considered the FFT core provided by Altera, and such FFT of 65 536 points using a streaming data flow and 16 bits of resolution implemented on an Altera Stratix V FPGA requires about 12.5 Mbit of memory [32]. The memory required by an Altera FFT roughly doubles when the length is doubled [34]; therefore we can assume that if it would exist, an FFT of 1 048 576 points would require approximately 200 Mbit. Note that nonetheless these amount of memory could be significatively reduced (by about 75 %) by using an alternative implementation of the circular correlation [32], although not reported in Tables III and IV.

Note also that to store the incoming signal (see Fig. 1), an additional memory is needed, for example of $2N_P N_S \times B = 1\,636\,800$ bits if $B = 2$ bits are used for the quantization.

In Tables III and IV, we clearly see that the pre-FFT and post-FFT sequential implementations require much less memory than the other implementations, but they have a much longer processing time. It can also be seen that when there is no zero-padding (high-end receiver case), the post-FFT implementation has the same processing time as the pre-FFT one but uses less memory. Still considering a sequential implementation, the use of a memory to store the correlation results increases a lot the memory for a small decrease of the processing time. Finally, the parallel implementations use a lot of memory but decrease a lot the processing time, and the post-FFT parallel implementations are better than the pre-FFT parallel implementation since the processing time and the memory can both be lower. The parallel FFT implementations have a processing time close to the one of the theoretical direct correlation, or even better for the high-end receiver (because

TABLE I
SUMMARY OF THE MEMORY REQUIREMENT AND PROCESSING TIME FOR THE DIFFERENT IMPLEMENTATIONS. *SCCC STANDS FOR SECONDARY CODE CIRCULAR CORRELATION.

Secondary code removal	Implementation	Memory for storage and combinations (bit)	Processing time (clock cycle)
	Theoretical direct correlation (Figs. 2 & 14)	-	$N(K+2)+2L_N$ or $N(2K+4)+N_{Z,D}(K+2)+2L_N$
Pre-FFT	Sequential (Figs. 3 & 15)	$4N_P(B_0+\lceil\log_2 N_S\rceil)$	$N_P\left(2KN_S^2+3\right)+2N_Z+2L$
	Parallel (Figs. 4 & 16)	$4N_PN_S(B_0+\lceil\log_2 N_S\rceil)$	$N_P[4K(N_S-1)+5]+N_Z[K(N_S-1)+2]+2L$
Post-FFT	Sequential (Figs. 6 & 19)	$2N_P(B_1+\lceil\log_2 N_S\rceil)$	$N_P\left(2KN_S^2+3\right)+N_Z\left(KN_S^2+1\right)+2L$
	Parallel (Figs. 7 & 20)	$2N_PN_S(B_1+\lceil\log_2 N_S\rceil)$	$N_P(2KN_S+3)+N_Z(KN_S+1)+2L$
	Memory + Sequential (Figs. 5 & 18)	$2N_P[(N_S+1)B_1+\lceil\log_2 N_S\rceil]$	$N_P\left[K\left(N_S^2-1\right)+N_S+4\right]+N_Z[K(N_S-1)+2]+2L$
	Memory + Sequential SCCC* (Figs. 8 & 21)	$2N_PN_SB_1$	$N_P\left[K\left(N_S^2+2N_S-2\right)+4\right]+N_Z[K(N_S-1)+2]+2L$
	Memory + Parallel SCCC (Figs. 9 & 22)	$2N_PN_SB_1$	$N_P[K(3N_S-2)+4]+N_Z[K(N_S-1)+2]+2L$
	Memory + FFT SCCC (Figs. 10 & 23)	$2N_PN_SB_1$	$N_P\left[K\left(4N_S+N_{Z,S}-2\right)+4\right]+N_Z[K(N_S-1)+2]+2L+2L_S+N_S$

TABLE II
PARAMETERS SELECTED FOR A "LOW-COST" AND A "HIGH-END" RECEIVER.

Parameters	Low-cost	High-end
B_0	4	12
f_S	20.48 MHz	32.768 MHz
N_P	20 480	32 768
N_{FFT}	65 536	65 536
$N_Z=N_{FFT}-2N_P$	24 576	0
L	0	0
N_S	20	20
$N_{FFT,S}$	64	64
$N_{Z,S}=N_{FFT,S}-2N_S$	24	24
B_1	16	16
K	100	100
L_S	0	0
N	409 600	655 360
$N_{Z,D}$	229 376	786 432
L_N	0	0

the FFT for the direct correlation uses 2^{21} points due to the chosen sampling frequency), whereas the direct correlation requires a much higher amount of memory for the very large FFTs (and a higher amount of logic, not mentioned in the tables). In conclusion, we can say that with both receivers, the most suitable implementations are post-FFT parallel implementations. And comparing both receivers, the high-end one uses more memory and the processing time is longer due to the higher quantization and sampling frequency. Of course, the sequential and parallel implementations considered here are the two extremes; it is also possible to test only few delays for the secondary code in parallel, which would balance the memory requirements and the processing time.

H. Use of dual read access memory

In the previous discussions, it was assumed that only one sample could be read from a memory at each clock cycle. However, the memories inside FPGAs usually propose a dual read access, and thus it is possible to read simultaneously two samples stored at different addresses. This can be used to improve the processing time of the implementations discussed previously, but not all of them can benefit from it, as discussed next.

For Fig. 3, if we can access two samples of $x_{i,n}$ at the same time, the processing time can be halved since the bottleneck is in the access of the input signal. However, since $x_{i,n}$ is after the mixer with the local carrier, it would require two local carrier generators, therefore it is not so straightforward to implement. For Fig. 4, the processing time can be reduced only a little bit, at most by a factor $4/3$ because the bottleneck is on the correlation computation, with the same complexity as before. For Figs. 5 and 8, the processing time can be almost halved since the bottleneck is mainly related to the memory reading, and it is simple to implement since it is related to the memory storing the correlation results and does not complicate the access to $x_{i,n}$. Fig. 9, the processing time can be reduced only a little bit, at most by a factor $6/5$ because the bottleneck is mostly on the correlation computation, with the same simplicity as previously. For the other implementations (Figs. 6, 7 and 10), having a double read access cannot be exploited and thus the processing time will stay the same.

IV. NEW METHOD TO REDUCE THE PROCESSING TIME

In this section, we describe a method that reduces the theoretical number of operations related to the secondary code correlation by about 50 %, and discuss its application for a hardware implementation. Note that this method is not an approximation, i.e. the output will be exactly the same as previously, and thus the performance in terms of sensitivity is exactly the same.

The main idea is to rewrite the local secondary code as

$$\begin{aligned}\mathbf{s}&=(\mathbf{s}-\mathbf{1})+\mathbf{1}\\&=\mathbf{s}'+\mathbf{1},\end{aligned}\tag{12}$$

where $\mathbf{1}$ is a vector composed of ones only. In this case, the elements of $\mathbf{s}'$ can have as value 0 or -2. Note that the local secondary code is not modified, it is simply expressed as

TABLE III
NUMERICAL APPLICATION WITH THE L5 PILOT SIGNAL FOR A "LOW-COST" RECEIVER. * THESE VALUES ARE FOR UNOPTIMIZED FFT IMPLEMENTATIONS AND COULD BE REDUCED BY ABOUT 75 % [32], SEE SECTION III-G FOR MORE DETAILS.

Secondary code removal	Implementation	Memory for FFTs* (Mbit)	Memory for storage and combinations (bit)	Processing time (clock cycle)
	Theoretical direct correlation (Figs. 2 & 14)	200	-	$4080N_P + \quad 102N_{Z,D} = 106\,954\,752$
Pre-FFT	Sequential (Figs. 3 & 15)	37.5	$36N_P = \quad 737\,280$	$80\,003N_P + \quad 2N_Z = 1\,638\,510\,592$
	Parallel (Figs. 4 & 16)	37.5	$720N_P = 14\,745\,600$	$7605N_P + \quad 1902N_Z = \quad 202\,493\,952$
Post-FFT	Sequential (Figs. 6 & 19)	37.5	$42N_P = \quad 860\,160$	$80\,003N_P + 40\,001N_Z = 2\,621\,526\,016$
	Parallel (Figs. 7 & 20)	37.5	$840N_P = 17\,203\,200$	$4003N_P + \quad 2001N_Z = \quad 131\,158\,016$
	Memory + Sequential (Figs. 5 & 18)	37.5	$682N_P = 13\,967\,360$	$39\,924N_P + \quad 1902N_Z = \quad 864\,387\,072$
	Memory + Sequential SCCC* (Figs. 8 & 21)	37.5	$640N_P = 13\,107\,200$	$43\,804N_P + \quad 1902N_Z = \quad 943\,849\,472$
	Memory + Parallel SCCC (Figs. 9 & 22)	37.5	$640N_P = 13\,107\,200$	$5804N_P + \quad 1902N_Z = \quad 165\,609\,472$
	Memory + FFT SCCC (Figs. 10 & 23)	37.5	$640N_P = 13\,107\,200$	$10\,204N_P + \quad 1902N_Z = \quad 255\,721\,492$

TABLE IV
NUMERICAL APPLICATION WITH THE L5 PILOT SIGNAL FOR "HIGH-END" RECEIVER. * THESE VALUES ARE FOR UNOPTIMIZED FFT IMPLEMENTATIONS AND COULD BE REDUCED BY ABOUT 75 % [32], SEE SECTION III-G FOR MORE DETAILS.

Secondary code removal	Implementation	Memory for FFTs* (Mbit)	Memory for storage and combinations (bit)	Processing time (clock cycle)
	Theoretical direct correlation (Figs. 2 & 14)	400	-	$4080N_P + \quad 102N_{Z,D} = 213\,909\,504$
Pre-FFT	Sequential (Figs. 3 & 15)	37.5	$68N_P = \quad 2\,228\,224$	$80\,003N_P + \quad 2N_Z = 2\,621\,538\,304$
	Parallel (Figs. 4 & 16)	37.5	$1360N_P = 44\,564\,480$	$7605N_P + \quad 1902N_Z = \quad 249\,200\,640$
Post-FFT	Sequential (Figs. 6 & 19)	37.5	$42N_P = \quad 1\,376\,256$	$80\,003N_P + 40\,001N_Z = 2\,621\,538\,304$
	Parallel (Figs. 7 & 20)	37.5	$840N_P = 27\,525\,120$	$4003N_P + \quad 2001N_Z = \quad 131\,170\,304$
	Memory + Sequential (Figs. 5 & 18)	37.5	$682N_P = 22\,347\,776$	$39\,924N_P + \quad 1902N_Z = 1\,308\,229\,632$
	Memory + Sequential SCCC* (Figs. 8 & 21)	37.5	$640N_P = 20\,971\,520$	$43\,804N_P + \quad 1902N_Z = 1\,435\,369\,472$
	Memory + Parallel SCCC (Figs. 9 & 22)	37.5	$640N_P = 20\,971\,520$	$5804N_P + \quad 1902N_Z = \quad 190\,185\,472$
	Memory + FFT SCCC (Figs. 10 & 23)	37.5	$640N_P = 20\,971\,520$	$10\,204N_P + \quad 1902N_Z = \quad 334\,364\,692$

the sum of two codes, and this concerns only the local code, not the incoming one. Thus, (8) and (9) can respectively be rewritten as

$$\begin{aligned}\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \mathbf{y}_2\\ \mathbf{y}_3\end{bmatrix} &= \begin{bmatrix}\mathbf{P}_T & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{P}_T & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{P}_T & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}_T\end{bmatrix}\\ &\quad \begin{bmatrix}s'_0+1 & s'_1+1 & s'_2+1 & s'_3+1\\ s'_3+1 & s'_0+1 & s'_1+1 & s'_2+1\\ s'_2+1 & s'_3+1 & s'_0+1 & s'_1+1\\ s'_1+1 & s'_2+1 & s'_3+1 & s'_0+1\end{bmatrix}\begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\\ \mathbf{x}_2\\ \mathbf{x}_3\end{bmatrix} \quad (13)\\ &= \begin{bmatrix}\mathbf{P}_T & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{P}_T & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{P}_T & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}_T\end{bmatrix}\left(\begin{bmatrix}\mathbf{a}'_0\\ \mathbf{a}'_1\\ \mathbf{a}'_2\\ \mathbf{a}'_3\end{bmatrix}+\begin{bmatrix}\mathbf{x}_\Sigma\\ \mathbf{x}_\Sigma\\ \mathbf{x}_\Sigma\\ \mathbf{x}_\Sigma\end{bmatrix}\right),\end{aligned}$$

and

$$\begin{aligned}\begin{bmatrix}\mathbf{y}_0\\ \mathbf{y}_1\\ \mathbf{y}_2\\ \mathbf{y}_3\end{bmatrix} &= \begin{bmatrix}s'_0+1 & s'_1+1 & s'_2+1 & s'_3+1\\ s'_3+1 & s'_0+1 & s'_1+1 & s'_2+1\\ s'_2+1 & s'_3+1 & s'_0+1 & s'_1+1\\ s'_1+1 & s'_2+1 & s'_3+1 & s'_0+1\end{bmatrix}\begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \mathbf{r}_2\\ \mathbf{r}_3\end{bmatrix} \quad (14)\\ &= \begin{bmatrix}s'_0 & s'_1 & s'_2 & s'_3\\ s'_3 & s'_0 & s'_1 & s'_2\\ s'_2 & s'_3 & s'_0 & s'_1\\ s'_1 & s'_2 & s'_3 & s'_0\end{bmatrix}\begin{bmatrix}\mathbf{r}_0\\ \mathbf{r}_1\\ \mathbf{r}_2\\ \mathbf{r}_3\end{bmatrix}+\begin{bmatrix}\mathbf{r}_\Sigma\\ \mathbf{r}_\Sigma\\ \mathbf{r}_\Sigma\\ \mathbf{r}_\Sigma\end{bmatrix},\end{aligned}$$

with $\mathbf{a}'_j = \sum_{i=0}^{N_S-1} s'_{((i-j))}\mathbf{x}_i$, $\mathbf{x}_\Sigma = \sum_{i=0}^{N_S-1}\mathbf{x}_i$ and $\mathbf{r}_\Sigma = \sum_{i=0}^{N_S-1}\mathbf{r}_i$. Note that (13) and (14) are not approximations of (8) and (9), the output $\mathbf{y}$ is exactly the same in all the cases. Only the way to compute $\mathbf{y}$ is different. Since $\mathbf{x}_\Sigma$ and $\mathbf{r}_\Sigma$ are the sum of signals still containing a secondary code, one may think that they contain mostly noise and thus that they are not useful and could be removed from the computation, but this would be a wrong idea. Even if they indeed contain mostly noise, these are simply intermediate results, and the noises present will be subtracted to the same noises when adding $\mathbf{x}_\Sigma$ and $\mathbf{a}'_i$ or $\mathbf{r}_\Sigma$ and the combinations of $\mathbf{r}_i$, and at the end the output $\mathbf{y}$ will have the same noise component as with the traditional method. Removing $\mathbf{x}_\Sigma$ or $\mathbf{r}_\Sigma$ from (13) and (14) would change the operation done, add more noise, and therefore impact the sensitivity. Therefore, (13) and (14) should be applied as it is.

In (8), the computation of one combination requires $(N_S-1)\,2N_P$ additions, thus the computation of the N_S combinations requires $N_S\,(N_S-1)\,2N_P = \left(N_S^2-N_S\right)2N_P$ additions. In (13), the computation of $\mathbf{x}_\Sigma$ requires $(N_S-1)\,2N_P$ additions, and then for each output $\mathbf{y}_k$, the computation of one combination $\mathbf{a}'_i$ requires $(N_S/2-1)\,2N_P$ additions in average (i.e. if half of the samples of $\mathbf{s}'$ are zeros), and the addition of $\mathbf{a}'_i$ and $\mathbf{x}_\Sigma$ requires $1\times 2N_P$ additions. Thus, the total number of operations for the N_S outputs $\mathbf{y}_k$ is $(N_S-1+N_S\,(N_S/2-1+1))\,2N_P = \left(N_S^2/2+N_S-1\right)2N_P$. Table V shows the number of additions of both equations considering 50 % of zeros in $\mathbf{s}'$ and for

TABLE V
NUMBER OF ADDITIONS OF VECTOR OF $2N_P$ POINTS FOR (8) AND (13), IN THE WORST CASE (50 % OF ZEROS IN $\mathbf{s}'$), AND IN THE GNSS CASE (60 % OF ZEROS IN $\mathbf{s}'$ FOR L5 AND E1 CODES, 53.44 % OF ZEROS IN $\mathbf{s}'$ IN AVERAGE FOR E5 CODES).

N_S	**for** (8)	**worst case**		**GNSS case**	
		for (13)	**reduction**	**for** (13)	**reduction**
4	12	11	8.3 %	-	-
20	380	219	42.4 %	179	52.9 %
25	600	336.5	43.9 %	274	54.3 %
100	9900	5099	48.5 %	4755	52.0 %

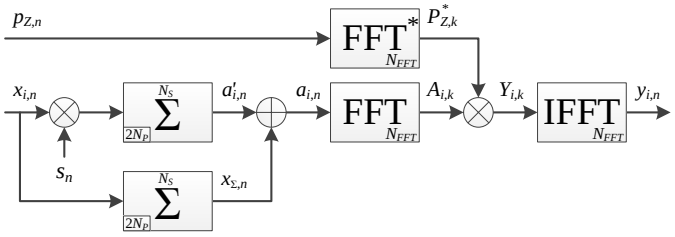

Fig. 11. Implementation of the pre-FFT secondary code removal using the proposed technique (Eq. (13)) computing each combination of the input sequentially. See details in Section IV, and the timing diagram in Fig. 24.

the actual number of zeros with the GNSS secondary codes. It can be seen that when N_S increases, the reduction of the number of operations approaches 50 % in the worst case, and it is slightly above 50 % for the GNSS signals. The same reduction is obtained for the post-FFT equation. Therefore, since this method reduces the number of operations, it can be useful for digital signal processor based receivers for example.

Now let's see the applicability for FPGA based receivers. For this, we will focus on the pre-FFT sequential implementation and (13). Previously, with (8), for each portion of the output ($\mathbf{y}_0$, $\mathbf{y}_1$, ...), it was necessary to combine N_S portions of the incoming code ($\mathbf{x}_0$, $\mathbf{x}_1$, ...) before performing one FFT-based correlation, as already shown in Fig. 15. Now, with (13), for each portion of the output ($\mathbf{y}_0$, $\mathbf{y}_1$, ...), it is necessary to combine only about half of the portions of the incoming code ($\mathbf{x}_0$, $\mathbf{x}_1$, ...) since in average half of the samples of s'_n are zero. Therefore, if a portion of the incoming code is multiplied by 0, we simply do not read it from the memory, and therefore the reading of the memory is about twice faster. However, we also need to add a special combination of the incoming code ($\mathbf{x}_\Sigma$, the sum of all the portions). But since this special combination is identical for all the portions of the output, we can compute it only once and store it into another memory. This memory will then be read when we will want to add $\mathbf{x}_\Sigma$ and $\mathbf{a}'_i$. Therefore accessing this second memory does not impact the processing time, because it is read simultaneously to the last $\mathbf{x}_i$ used to compute $\mathbf{a}'_i$, as shown in Fig. 24. The corresponding implementation is shown in Fig. 11.

For example, if we consider that $\mathbf{s} = \begin{bmatrix}-1 & 1 & 1 & -1\end{bmatrix}^T$, then $\mathbf{s}' = \begin{bmatrix}-2 & 0 & 0 & -2\end{bmatrix}^T$, and the combinations of the portions of the incoming code become

$$\begin{bmatrix}\mathbf{a}'_0\\ \mathbf{a}'_1\\ \mathbf{a}'_2\\ \mathbf{a}'_3\end{bmatrix} = \begin{bmatrix}-2 & 0 & 0 & -2\\ -2 & -2 & 0 & 0\\ 0 & -2 & -2 & 0\\ 0 & 0 & -2 & -2\end{bmatrix}\begin{bmatrix}\mathbf{x}_0\\ \mathbf{x}_1\\ \mathbf{x}_2\\ \mathbf{x}_3\end{bmatrix}. \quad (15)$$

Therefore, to compute each portion of the output ($\mathbf{y}_0$, $\mathbf{y}_1$, ...), it is necessary to read only two portions of the incoming code ($\mathbf{x}_0$, $\mathbf{x}_1$, ...) instead of four, as illustrated in Fig. 24. The processing starts by accessing all the portions of the input ($\mathbf{x}_0$, $\mathbf{x}_1$, ...) and summing them to compute and store $\mathbf{x}_\Sigma$. Then, it works as the pre-FFT sequential implementation except that only the portions of the input that are not multiplied by zero are accessed, and that $\mathbf{x}_\Sigma$ will be added when each $\mathbf{a}'_i$ will be available.

With this implementation, the memory needed is twice $4N_P\left(B_0 + \lceil \log_2 N_S \rceil\right)$ bits for the accumulation, as for the pre-FFT parallel implementation using two accumulators. However, looking at the processing time of both implementations (Figs. 17 and 24), the one using the new method can have a lower processing time because it is possible than more than half of the sample of $\mathbf{s}'$ are zeros, and because the zero-padding has less impact.

For example, the L5 pilot secondary code contains 12 ones and 8 minus ones. Therefore, the code $\mathbf{s}'$ will contain 12 zeros, i.e. 60 % of the total length. Making the same numerical application as in Section III-G with the "low-cost" receiver, the memory needed for both implementations is $72N_P = 1\,474\,560$ bits, and the processing time is $36\,003N_P + 2N_Z = 737\,390\,592$ clock cycles for the pre-FFT sequential implementation using the new method, and $40\,005N_P + 1002N_Z = 843\,927\,552$ clock cycles for the pre-FFT parallel implementation using two accumulators, which means a reduction of about 12.6 %. Therefore, the use of the proposed technique may be interesting for a hardware implementation. Note that the use of double read access can be exploited to approximately halve the processing time.

Of course, the choice of subtracting or adding one to the secondary code in (12) depends on the code that we have. The goal is to have as many zeros as possible in $\mathbf{s}'$. Note that there are also some variants of this method providing better performance but not applicable to every code [35].

V. CONCLUSION

In this paper, we have performed a comparison of the possible hardware implementations of the parallel code search acquisition for GNSS signals having a secondary code. Since applying directly the FFT over the entire tiered code is not possible or at least extremely consuming in hardware, a better solution already suggested in the literature is to perform FFT-based circular correlations over the primary code and to combine the results. Focusing on this solution, we have compared different hardware implementations, including the cases when the combinations are performed before or after the FFT-based correlations; when they are performed sequentially or in parallel, and when the output is provided in different orders. Moreover, we also analyzed the memory requirements and the processing time of each implementation.

From these comparisons, it has been shown that some implementations are not interesting (such as the one using a second FFT-based circular correlation for the secondary code), since they consume more memory and provide a longer processing time than other implementations. It has also been shown that the direct correlation that applies the FFT over the entire tiered code would not be interesting, because some proposed parallel implementations provides slightly longer processing times, but require much less memory.

Generally, the choice of the most suitable implementation is a compromise between the memory used and the processing time. However, if the various parameters are specified (such as the quantization of the signals, the sampling frequency, the number of coherent or non-coherent accumulation, the number of frequency bins to test), it will be easy to evaluate both the memory and the processing time using our results since all the formulas are provided.

In addition, we also have proposed a new method that approximately halves the number of operations related to the secondary code correlation, and slightly reduces the total processing time (12.6 %) for a hardware implementation. The idea of this method (which is not an approximation) is to add or subtract 1 to the binary secondary code to obtain a code with at least half of zeros to perform the correlation.

Appendix A
Memory-based accumulator

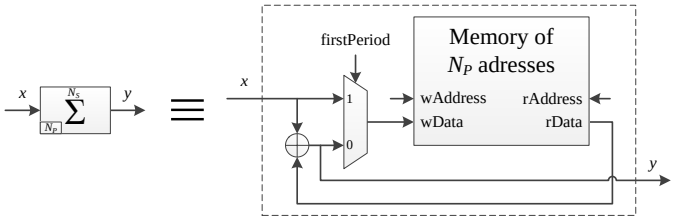

Fig. 12. Memory-based accumulator, summing N_S sequences of N_P samples.

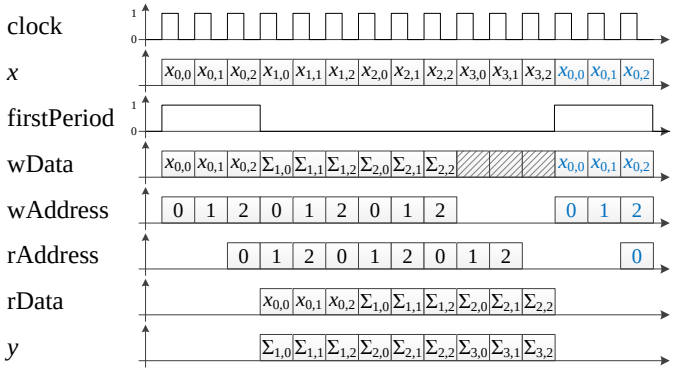

Fig. 13. Timing diagram of Fig. 5, with $N_P = 3$ and $N_S = 4$. The notation $\sum_{s,p}$ corresponds to $\sum_{k=0}^{s} x_{k,p}$.

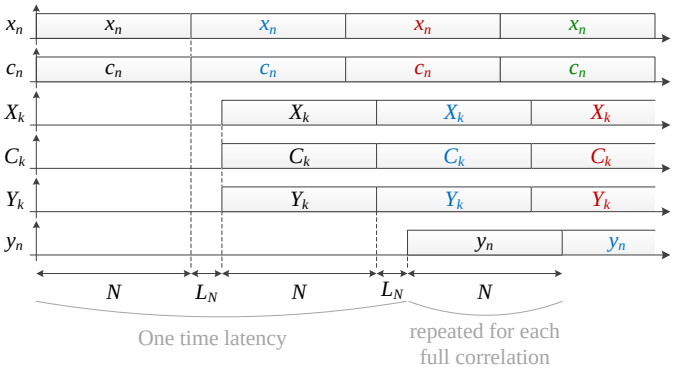

Fig. 14. Timing diagram of the implementation of the direct correlation over the secondary code period (Fig. 2). The colors indicate successive periods, which may be used for further coherent or non-coherent integration or for testing other carrier frequencies. The processing time to compute K times the full correlation is $N+L_N+N+L_N+K\times N = N_P\,(KN_S+2N_S)+2L_N$.

Appendix B
Timing diagram of the implementations

References

[1] F. van Diggelen, *A-GPS: Assisted GPS, GNSS, and SBAS*, ser. GNSS Technology and Applications Series. Artech House, 2009.

[2] A. Broumandan, J. Nielsen, and G. Lachapelle, "Coherent integration time limit of a mobile receiver for indoor GNSS applications," *GPS Solutions*, vol. 16, no. 2, pp. 157–167, April 2012.

[3] T. Pany, B. Riedl, J. Winkel, T. Worz, R. Schweikert, H. Niedermeier, S. Lagrasta, G. Risueno, and D. Banos, "Coherent integration time: the longer, the better," *Inside GNSS*, vol. 4, no. 6, pp. 52–61, November/December 2009.

[4] P. Gaggero and D. Borio, "Ultra-stable oscillators: limits of GNSS coherent integration," in *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, Savannah, USA, September 2008, pp. 565–575.

[5] C. Macabiau, L. Ries, F. Bastide, and J.-L. Issler, "GPS L5 receiver implementation issues," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS/GNSS 2003)*, Portland, USA, September 2003, pp. 153–164.

[6] C. Mongrédien, G. Lachapelle, and M. Cannon, "Testing GPS L5 acquisition and tracking algorithms using a hardware simulator," in *Proceedings of the 19th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2006)*, Fort Worth, USA, September 2006, pp. 2901–2913.

[7] G. Corazza, C. Palestini, R. Pedone, and M. Villanti, "Galileo primary code acquisition based on multi-hypothesis secondary code ambiguity elimination," in *Proceedings of the 20th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2007)*, Fort Worth, USA, September 2007, pp. 2459–2465.

[8] D. Borio, "M-sequence and secondary code constraints for GNSS signal acquisition," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 2, pp. 928–945, April 2011.

[9] N. Shivaramaiah, A. Dempster, and C. Rizos, "Exploiting the secondary codes to improve signal acquisition performance in Galileo receivers," in *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, Savannah, USA, September 2008, pp. 1497–1506.

[10] Y. Tawk, A. Jovanovic, J. Leclère, C. Botteron, and P.-A. Farine, "A new FFT-based algorithm for secondary code acquisition for Galileo signals," in *IEEE Vehicular Technology Conference (VTC Fall)*, San Francisco, USA, September 2011, pp. 1–6.

[11] C. Hegarty, M. Tran, and A. Van Dierendonck, "Acquisition algorithms for the GPS L5 signal," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS/GNSS 2003)*, Portland, USA, September 2003, pp. 165–177.

[12] C. Yang, C. Hegarty, and M. Tran, "Acquisition of the GPS L5 signal using coherent combining of I5 and Q5," in *Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2004)*, Long Beach, USA, September 2004, pp. 2184–2195.

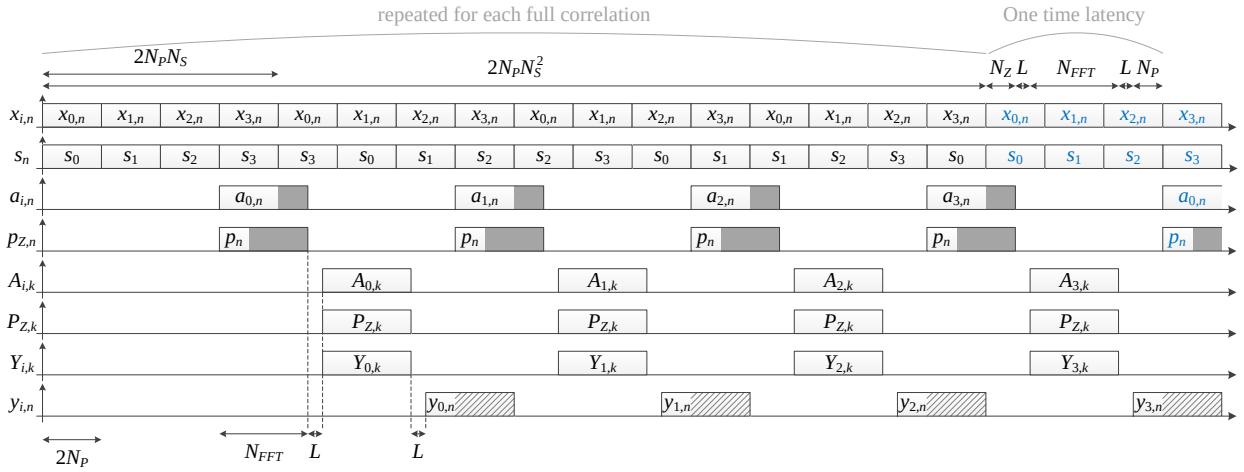

Fig. 15. Timing diagram of the implementation of the pre-FFT secondary code removal in a sequential way (Fig. 3) with $N_S = 4$. Grey parts indicate zeros (N_Z samples are padded to each accumulation result before the FFT, thus the FFTs length is $N_{FFT} = 2N_P + N_Z$), and hatched parts indicate we do not care about these samples. The processing time to compute K times the full correlation (i.e. K times N_S outputs y_i) is $K \times 2N_P N_S^2 + N_Z + L + N_{FFT} + L + N_P = N_P \left(2KN_S^2 + 3\right) + 2N_Z + 2L$.

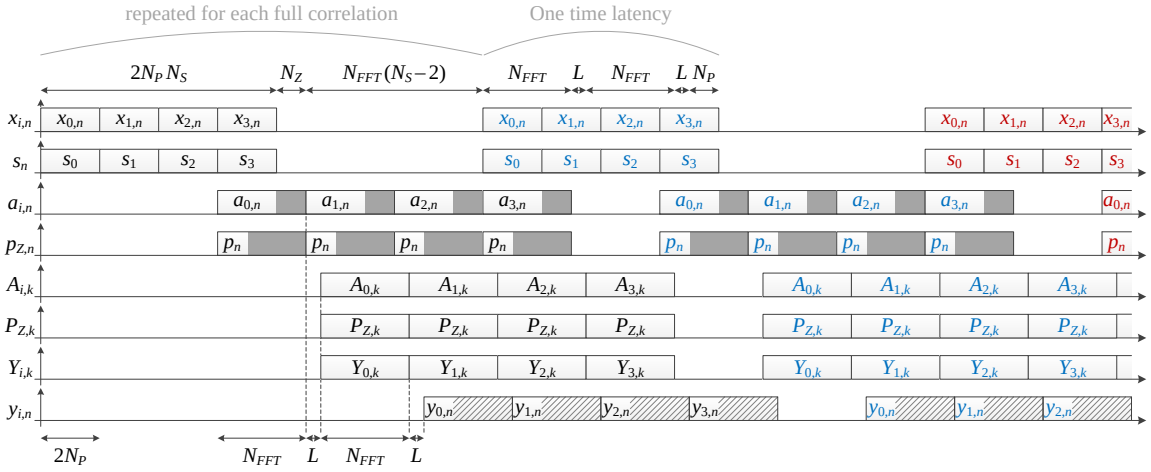

Fig. 16. Timing diagram of the implementation of the pre-FFT secondary code removal in a parallel way (Fig. 4) with $N_S = 4$. The processing time to compute K times the full correlation is $K \times [2N_P N_S + N_Z + N_{FFT}(N_S - 2)] + N_{FFT} + L + N_{FFT} + L + N_P = N_P [4K(N_S - 1) + 5] + N_Z [K(N_S - 1) + 2] + 2L$.

[13] J. Leclère, "Resource-efficient parallel acquisition architectures for modernized GNSS signals," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, Switzerland, 2014.

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

[15] D. M. Akos and M. Pini, "Effect of sampling frequency on GNSS receiver performance," *NAVIGATION, Journal of The Institute of Navigation*, vol. 53, no. 2, pp. 85–96, Summer 2006.

[16] K. Borre, D. Akos, N. Bertelsen, P. Rinder, and S. Jensen, *A software-defined GPS and Galileo receiver. Single-frequency approach*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2007.

[17] U. Cheng, W. Hurd, and J. Statman, "Spread-spectrum code acquisition in the presence of Doppler shift and data modulation," *IEEE Transactions on Communications*, vol. 38, no. 2, pp. 241–250, February 1990.

[18] S. Spangenberg and G. Povey, "Code acquisition for LEO satellite mobile communication using a serial-parallel correlator with FFT for Doppler estimation," in *International Symposium on Communication Systems and Digital Signal Processing (CSDSP)*, Sheffield, UK, April 1998.

[19] H. Mathis, P. Flammant, and A. Thiel, "An analytic way to optimize the detector of a post-correlation FFT acquisition algorithm," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS/GNSS 2003)*, Portland, USA, September 2003, pp. 689–699.

[20] D. van Nee and A. Coenen, "New fast GPS code-acquisition technique using FFT," *Electronics Letters*, vol. 27, no. 2, pp. 158–160, Jan 1991.

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

[22] N. Ziedan, *GNSS receivers for weak signals*, ser. GNSS Technology and Applications Series. Artech House, 2006.

[23] M. Foucras, O. Julien, C. Macabiau, and B. Ekambi, "A novel computationally efficient Galileo E1 OS acquisition method for GNSS software receiver," in *Proceedings of the 25th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2012)*, Nashville, USA, September 2012, pp. 365–383.

[24] J. Leclère, C. Botteron, and P.-A. Farine, "Comparison framework of FPGA-based GNSS signals acquisition architectures," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 3, pp. 1497–1518, July 2013.

[25] ——, "Expressing discrete convolutions and correlations using matrices and polynomials: a unified presentation," *IEEE Signal Processing Magazine*, submission in 2017.

[26] Altera, *FFT MegaCore Function User Guide*, August 2014.

[27] Xilinx, *LogiCORE IP Fast Fourier Transform Product Guide*, October 2014.

[28] Lattice, *FFT Compiler IP Core Users Guide*, August 2011.

[29] Microsemi, *Core FFT Handbook*, September 2013.

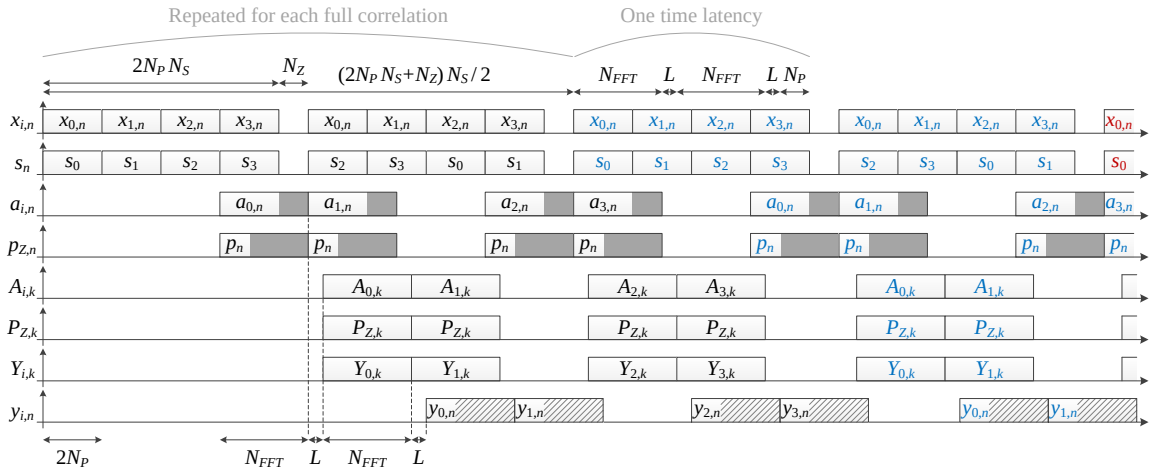

Fig. 17. Timing diagram of the implementation of the pre-FFT secondary code removal in a parallel way (Fig. 4 with only two accumulators) with $N_S = 4$. The processing time to compute K times the full correlation is $K \times \left[(2N_P N_S + N_Z)\frac{N_S}{2}\right] + N_{FFT} + L + N_{FFT} + L + N_P = N_P\left(KN_S^2 + 5\right) + N_Z\left(K\frac{N_S}{2} + 2\right) + 2L$.

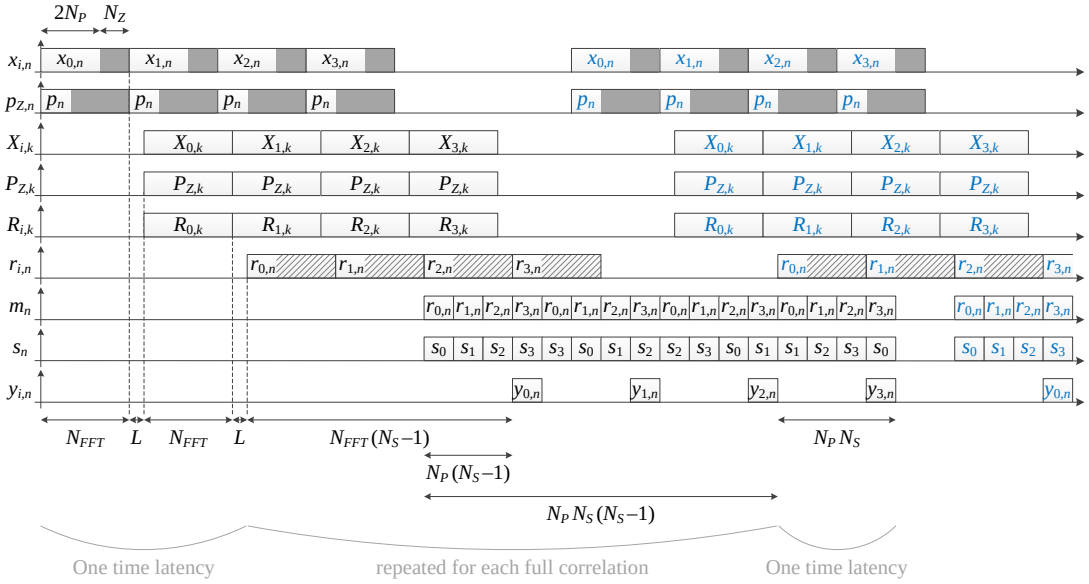

Fig. 18. Timing diagram of the implementation of the post-FFT secondary code removal using a memory (Fig. 5) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT} + L + N_{FFT} + L + K\left[N_{FFT}\left(N_S - 1\right) - N_P\left(N_S - 1\right) + N_P N_S\left(N_S - 1\right)\right] + N_P N_S = N_P\left[K\left(N_S^2 - 1\right) + N_S + 4\right] + N_Z\left[K\left(N_S - 1\right) + 2\right] + 2L$.

[30] J. Leclère, C. Botteron, and P.-A. Farine, "Resource and performance comparisons for different acquisition methods that can be applied to a VHDL-based GPS receiver in standalone and assisted cases," in *IEEE/ION Position Location and Navigation Symposium (PLANS)*, May 2010, pp. 745–751.

[31] ——, "Acquisition of modern GNSS signals using a modified parallel code-phase search architecture," *Signal Processing*, vol. 95, pp. 177–191, February 2014.

[32] J. Leclère, C. Botteron, R. Landry, and P.-A. Farine, "FFT splitting for improved FPGA-based acquisition of GNSS signals," *International Journal of Navigation and Observation*, vol. 2015, November 2015, article ID 765898.

[33] E. Kaplan and C. Hegarty, *Understanding GPS: Principles and applications*, 2nd ed., ser. GNSS Technology and Applications Series. Artech House, 2005.

[34] J. Leclère, C. Botteron, and P.-A. Farine, "Implementing super-efficient FFTs in Altera FPGAs," *EE Times Programmable Logic Designline*, February 2015, available online at www.eetimes.com, http://infoscience.epfl.ch/record/204540.

[35] J. Leclère and R. Landry, "Complexity reduction for high sensitivity acquisition of gnss signals with a secondary code," in *Proceedings of the 29th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2016*, Portland, OR, USA, September 2016, pp. 436–443.

Dr Jérôme Leclère received a master and 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 with the LASSENA, ÉTS, Montréal, Canada. He focuses his researches in the reduction of the complexity of the acquisition of GNSS signals, with application to hardware receivers, especially using FPGAs, and on the GNSS/INS integration. He developed an FPGA-based high sensitivity assisted GPS L1 C/A receiver, and participated to the design of several FPGA receivers, for space applications (L1 C/A) and for GNSS reflectometry (L1/E1).

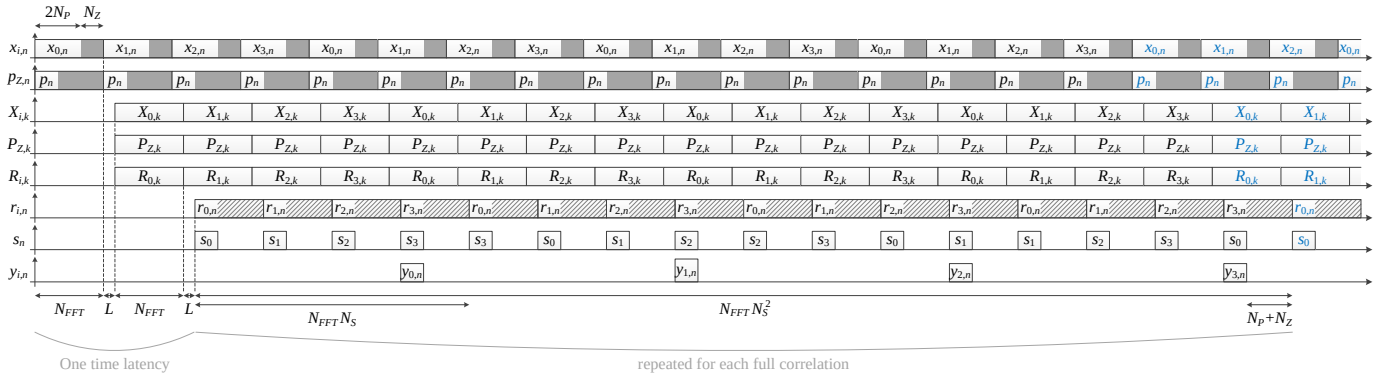

Fig. 19. Timing diagram of the implementation of the post-FFT secondary code removal without memory in a sequential way (Fig. 6) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT}+L+N_{FFT}+L+KN_{FFT}N_S^2-(N_P+N_Z) = N_P\left(2KN_S^2+3\right)+N_Z\left(KN_S^2+1\right)+2L$.

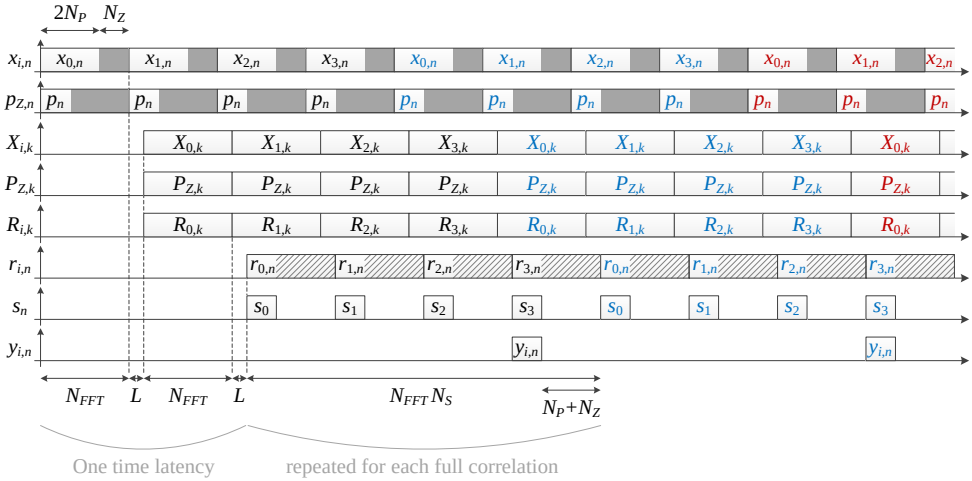

Fig. 20. Timing diagram of the implementation of the post-FFT secondary code removal without memory in a parallel way (Fig. 7) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT}+L+N_{FFT}+L+KN_{FFT}N_S-(N_P+N_Z) = N_P\left(2KN_S+3\right)+N_Z\left(KN_S+1\right)+2L$.

Dr Cyril Botteron is leading, managing, and coaching the research and project activities of the Global Navigation Satellite System and Ultra-Wideband and mm-wave groups at École Polytechnique Fédérale de Lausanne (EPFL). He is the author or co-author of 5 patents and over 80 publications in major journals and conferences in the fields of wireless positioning systems, GNSS-based navigation and sensing, ultra-low-power radio frequency communications and integrated circuits design, and baseband analog and digital signal processing.

Prof. Pierre-André Farine is professor in electronics and signal processing at EPFL, and is head of the electronics and signal processing laboratory. He received the M.Sc. and Ph.D. degrees in Micro technology from the University of Neuchâtel, Switzerland, in 1978 and 1984, respectively. He is active in the study and implementation of low-power solutions for applications covering wireless telecommunications, ultra-wideband, global navigation satellite systems, and video and audio processing. He is the author or co-author of more than 100 publications in conference and technical journals and 50 patent families (more than 270 patents).

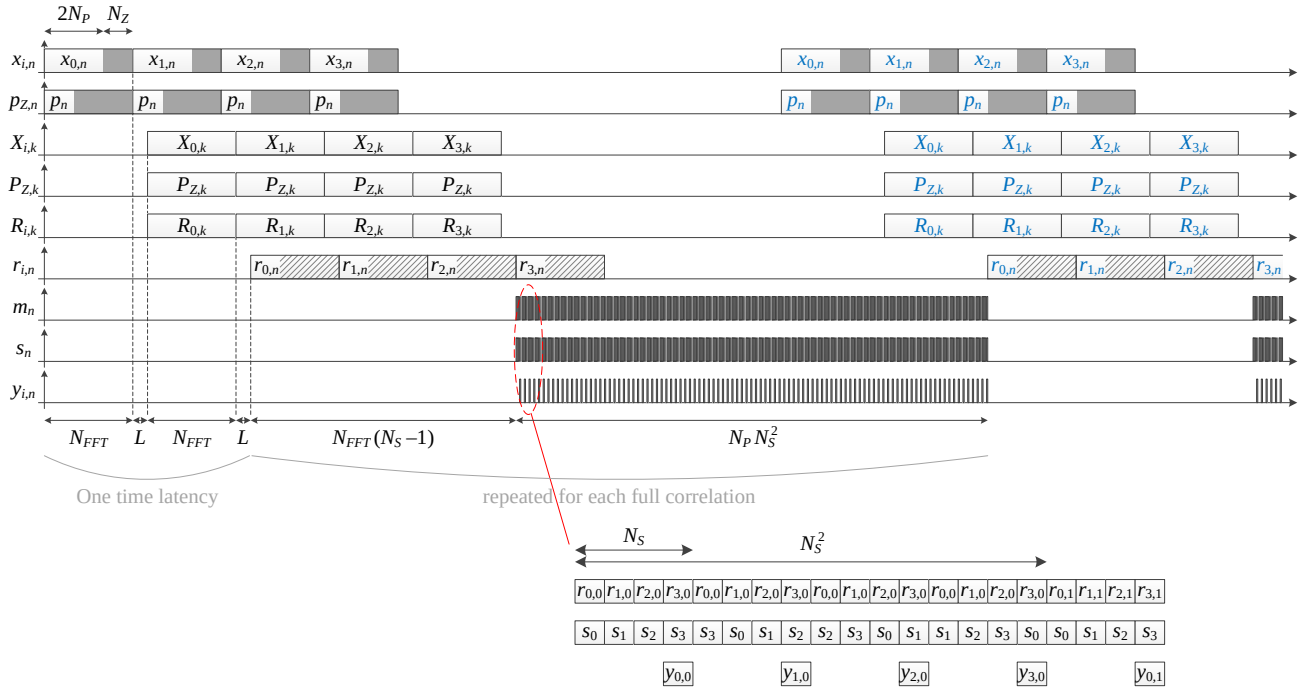

Fig. 21. Timing diagram of the implementation of the post-FFT secondary code removal using a memory followed by a sequential circular correlation (Fig. 8) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT} + L + N_{FFT} + L + K\left[N_{FFT}\left(N_S - 1\right) + N_P N_S^2\right] = N_P\left[K\left(N_S^2 + 2N_S - 2\right) + 4\right] + N_Z\left[K\left(N_S - 1\right) + 2\right] + 2L$.

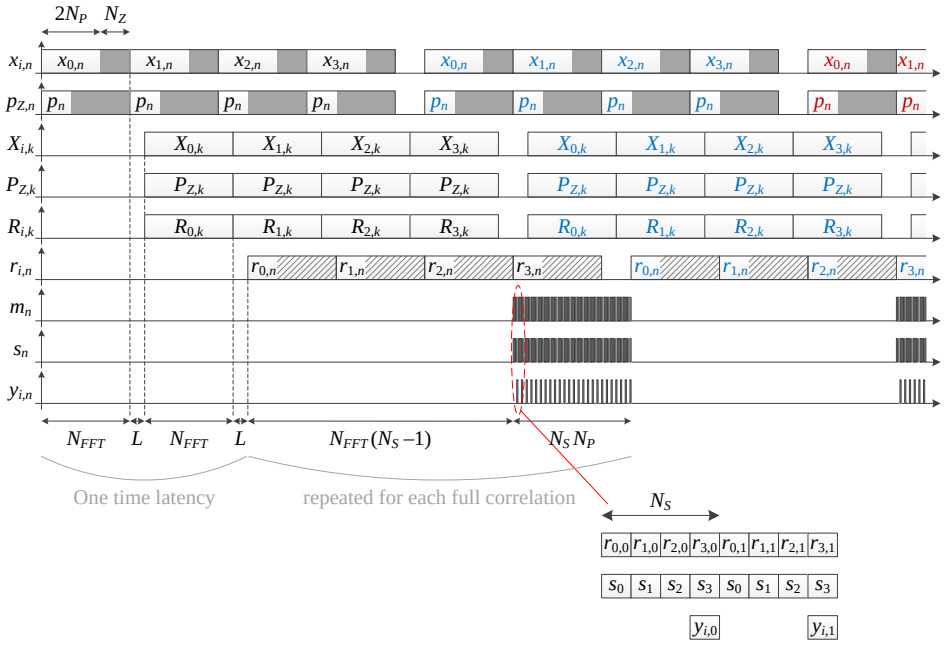

Fig. 22. Timing diagram of the implementation of the post-FFT secondary code removal using a memory followed by a parallel circular correlation (Fig. 9) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT} + L + N_{FFT} + L + K\left[N_{FFT}\left(N_S - 1\right) + N_S N_P\right] = N_P\left[K(3N_S - 2) + 4\right] + N_Z\left[K\left(N_S - 1\right) + 2\right] + 2L$.

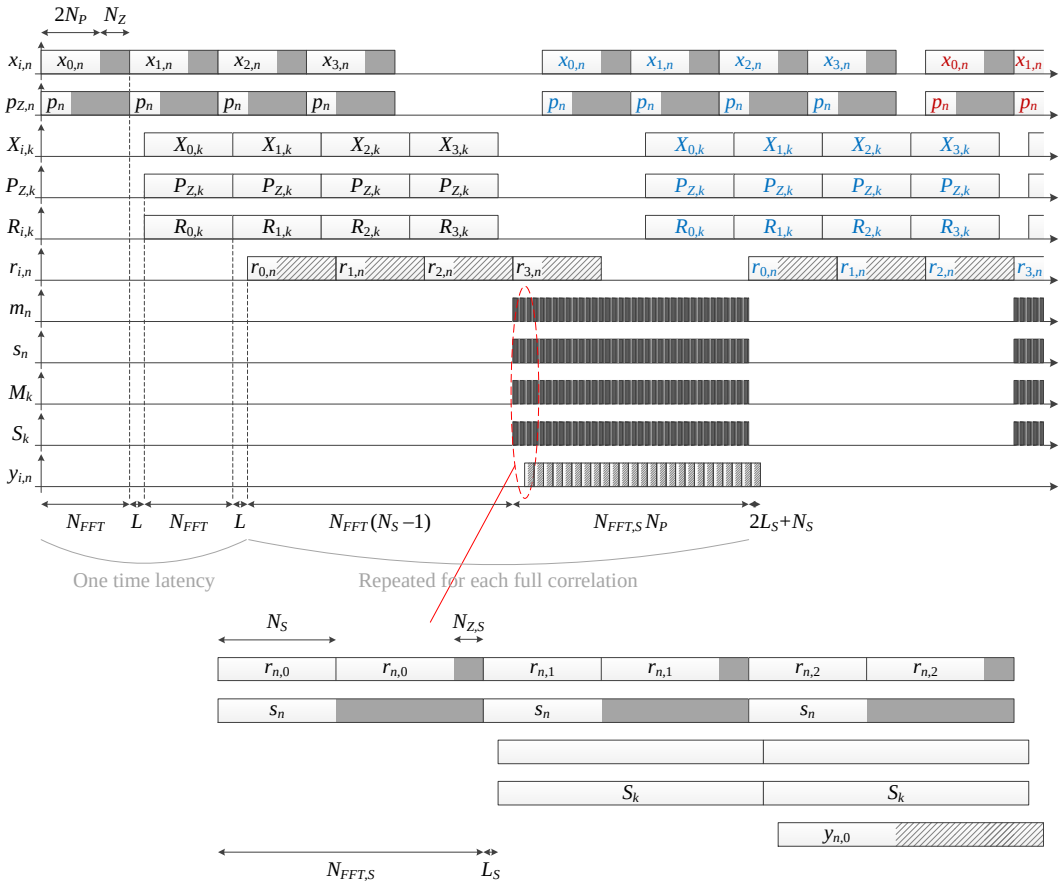

Fig. 23. Timing diagram of the implementation of the post-FFT secondary code removal using a memory followed by an FFT-based circular correlation (Fig. 10) with $N_S = 4$. The processing time to compute K times the full correlation is $N_{FFT}+L+N_{FFT}+L+K\left[N_{FFT}\left(N_S-1\right)+N_{FFT_S}N_P\right]+2L_S+N_S = N_P\left[K\left(4N_S+N_{Z,S}-2\right)+4\right]+N_Z\left[K\left(N_S-1\right)+2\right]+2L+2L_S+N_S$.

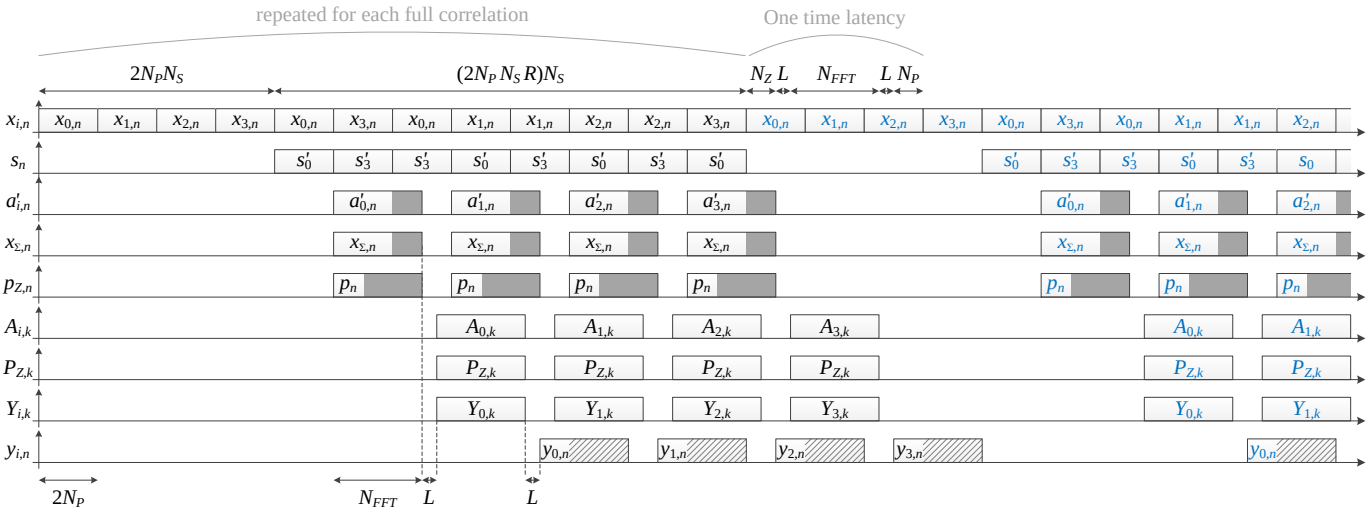

Fig. 24. Timing diagram of the implementation of the pre-FFT secondary code removal in a sequential way using the new technique for the combinations (Fig. 11) with $N_S = 4$. R denotes the ratio between the number of non zero values in $\mathbf{s}'$ and N_S ($R = 0.5$ here). The processing time to compute K times the full correlation is $K \times \left[2N_P N_S + \left(2N_P N_S R\right) N_S\right] + N_Z + L + N_{FFT} + L + N_P = N_P\left[2K\left(N_S^2 R + N_S\right)+3\right] + 2N_Z + 2L$.