Hindawi Publishing Corporation
International Journal of Navigation and Observation
Volume 2015, Article ID 765898, 12 pages
http://dx.doi.org/10.1155/2015/765898

Hindawi

Research Article

FFT Splitting for Improved FPGA-Based Acquisition of GNSS Signals

Jérôme Leclère,[1,2] Cyril Botteron,[1] René Jr. Landry,[2] and Pierre-André Farine[1]

[1]*Electronics and Signal Processing Laboratory (ESPLAB), École Polytechnique Fédérale de Lausanne (EPFL), 2000 Neuchâtel, Switzerland*
[2]*Laboratory of Space Technologies, Embedded Systems, Navigation and Avionic (LASSENA), École de Technologie Supérieure (ÉTS), Montréal, QC, Canada H3C 1K3*

Correspondence should be addressed to Jérôme Leclère; jerome.leclere@lassena.etsmtl.ca

Received 18 September 2015; Accepted 25 November 2015

Academic Editor: Letizia Lo Presti

With modern global navigation satellite system (GNSS) signals, the FFT-based parallel code search acquisition must handle the frequent sign transitions due to the data or the secondary code. There is a straightforward solution to this problem, which consists in doubling the length of the FFTs, leading to a significant increase of the complexity. The authors already proposed a method to reduce the complexity without impairing the probability of detection. In particular, this led to a 50% memory reduction for an FPGA implementation. In this paper, the authors propose another approach, namely, the splitting of a large FFT into three or five smaller FFTs, providing better performances and higher flexibility. For an FPGA implementation, compared to the previously proposed approach, at the expense of a slight increase of the logic and multiplier resources, the splitting into three and five allows, respectively, a reduction of 40% and 64% of the memory, and of 25% and 37.5% of the processing time. Moreover, with the splitting into three FFTs, the algorithm is applicable for sampling frequencies up to 24.576 MHz for L5 band signals, against 21.846 MHz with the previously proposed algorithm. The algorithm is applied here to the GPS L5 and Galileo E5a, E5b, and E1 signals.

1. Introduction

The question of computing a circular correlation between a local code replica and an incoming code having a bit sign transition is a recurrent problem in global navigation satellite system (GNSS) [1–9]. This problem, already present with the GPS L1 C/A signal, is even more important with the modern GPS and Galileo signals, because of the higher data rate and the presence of the secondary code that imply a potential bit sign transition in each consecutive period of the primary code. This problem appears with the parallel code search (PCS) acquisition method, where a circular correlation is performed through fast Fourier transforms (FFTs) over one period of the primary code [4].

The straightforward solution to this problem is to at least double the length of the sequences, by using more samples of the input signal (to observe at least two code periods and thus to be sure to observe one code period that is free of sign transition) and by zero-padding the local code replica [4, 9]. However, this method implies using longer FFTs, which increases the processing complexity, and at least half of the calculated samples are discarded, making this solution suboptimal.

Note that this straightforward solution is also a solution to other problems. (1) Still with the PCS, the length of the sequences may need to be increased to satisfy a constraint on the FFT length. For example, if the FFT length must be a power of two and if one code period corresponds to 4000 samples, applying directly zero-padding on both incoming and local sequences to get sequences of 4096 samples will result in losses (in general, the zeros will not be inserted at the same position inside the received and local codes; see [10] for more details). To avoid these losses, at least two code periods should be observed in the incoming or local signal. In the previous example, the incoming sequence would thus be composed of 8000 signal samples padded with 192 zeros, whereas the local code would be composed of 4000 samples padded with 4192 zeros. (2) With another acquisition method

known as double block zero padding (DBZP) [11, 12], the incoming signal and the local code are cut in small portions and the correlations are computed on these small portions. However, in order to compute correctly the correlation, each portion of the local code is padded with zeros and correlated with two consecutive portions of the incoming signal, as in the straightforward solution.

In this paper, we propose the use of a method to reduce the zero-padding in order to improve the efficiency. The method is based on the fact that an N-point FFT can be computed using K N/K-point FFTs [13, Section 13.47]. In particular, we focus on the computation of an N-point FFT using three $N/3$-point FFTs or five $N/5$-point FFTs. This method is applied in other areas (this is used, e.g., in new mobile phone long term evolution technology, where 512-point FFTs are used to compute a 1536-point FFT [14]), but it has not yet been applied to the acquisition of GNSS signals.

The rest of the paper is organized as follows. Section 2 only briefly recalls the problem and the current solutions, since these were already detailed in [4]. Section 3 describes the FFT splitting algorithm. Section 4 evaluates the performance of the proposed algorithm for the acquisition of the L5, E5a, E5b, and E1 signals, in terms of complexity and resources for an FPGA implementation. Finally, Section 5 concludes this paper.

2. Problem and Current Solutions

The signal received from a GNSS satellite contains a spreading code, whose beginning is unknown, and an unknown residual carrier frequency due to the Doppler effect. The aim of the acquisition is to determine the code delay and the carrier frequency for all visible satellites [15]. Among different methods, there is the basic serial search (SS) [16], there are one-dimensional parallel searches such as the parallel frequency search (PFS) and the PCS [16], [17, Chap. 2], and there are two-dimensional parallel searches such as the DBZP method [11, 12] (which is an extension of the PFS) or the methods proposed in [18, 19]. The parallel code search method seems to be among the most attractive methods for the acquisition of modern signals, because it can compensate the code Doppler (especially important with signals having a high chipping rate and with weak signals requiring a long integration) whereas some other methods (such as those based on the PFS) cannot compensate it [20], and one-dimensional parallel methods are less complex to implement than two-dimensional parallel methods.

As mentioned in the introduction, the PCS performs a circular correlation between a local code replica of the satellite searched and the received signal using FFTs, usually over the primary code period. Then, extra coherent integration or noncoherent integration can be performed, as shown in Figure 1 (note that the FFT of the local code could be precomputed and stored in a memory to use only 2 FFTs in real time). If the Doppler frequency is correctly eliminated and if there is no sign transition, a peak will appear at the correct code delay, as illustrated in Figure 2(a). However, in the presence of a sign transition between two consecutive

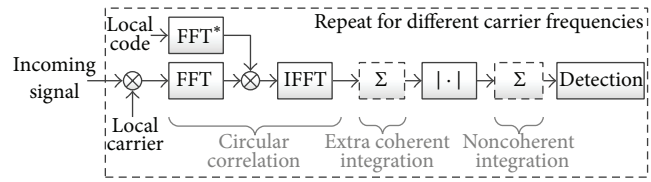

Figure 1: Illustration of the parallel code search (PCS) acquisition. The $*$ denotes the complex conjugate operation.

code periods (due to the data or the secondary code), there may be a loss depending on the position of this sign transition in the received signal, as shown in Figures 2(b) and 2(c). Nevertheless, note that when the correlation peak decreases at the correct Doppler frequency bin due to a sign transition, other peaks will appear with a lower amplitude at incorrect Doppler frequency bins [8].

2.1. Straightforward Solution (3-FFT Solution). As mentioned in the introduction, the straightforward solution to this problem is to at least double the length of the sequences, by using more samples of the input signal and by zero-padding the local code [9], as shown in Figure 3. In this case, we are sure to observe a complete period of the code in the received signal and thus to obtain a peak that is not attenuated. This peak is in the first half of the correlation output, while the second half contains a peak potentially attenuated (depending on the bit sign transitions). Therefore, the second half is usually discarded, which is not optimal. For the rest of this paper, this solution will be called the 3-FFT solution. As mentioned earlier, it would be possible to use only 2 FFTs in real time by precomputing the FFT of the local code and storing it in a memory, but this case is not studied here. However, it is relatively easy to adapt the results of Section 4 to this case.

The L5, E5a, and E5b signals have a spreading code period of 1 ms, and the usual minimum sampling frequency considered is 20.46 MHz (assuming a complex sampling since the signal bandwidth is 20.46 MHz) [4], which is twice the chipping rate (note that in practice this exact frequency is never used because of accuracy problems in tracking and positioning [21]). Therefore, in one code period there are at least 20 460 samples, and in two periods there are at least 40 920 samples. The E1 signal has a code period of 4 ms, and when this signal is processed as a BOC(1,1) signal with the PCS, the usual minimum sampling frequency considered is 6.138 MHz (still assuming a complex sampling since the signal bandwidth is 4.092 MHz) [4], which is six times the chipping rate (the ratio between the sampling frequency and the chipping rate is higher for BOC than BPSK modulation because the autocorrelation functions are different [22]). Therefore, in one period of the code there are at least 24 552 samples, and in two periods there are at least 49 104 samples. Thus, for all these signals, if the FFT length must be a power of two, the 3-FFT solution will use 65 536-point FFTs, for sampling frequencies between 20.46 and 32.768 MHz for the L5, E5a, and E5b signals and between 6.138 and 8.192 MHz for the E1 signal processed as BOC(1,1). If the sampling

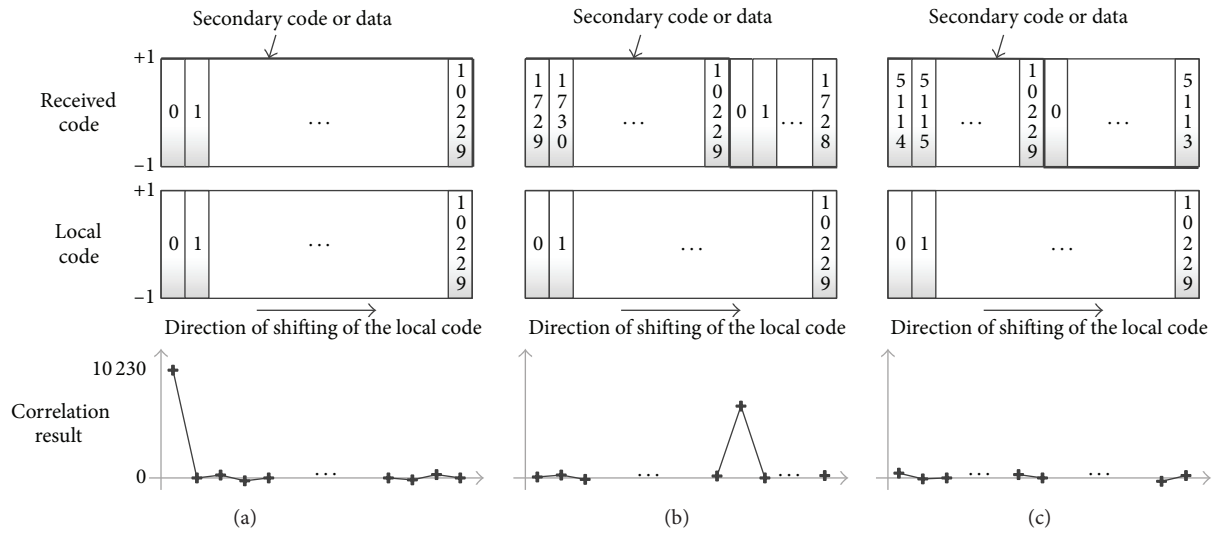

Figure 2: Illustration of the problem due to a transition coming from the secondary code or the data. The values in the boxes indicate the chip number. (a) The incoming primary code starts with the first chip; the correlation at the correct alignment is maximum, as it would be without data or secondary code. (b) The incoming primary code does not start at the first chip (usual case); the correlation at the correct alignment is reduced. (c) In the worst case, the incoming primary code starts at the middle of a period; the correlation at the correct alignment is very close to 0.

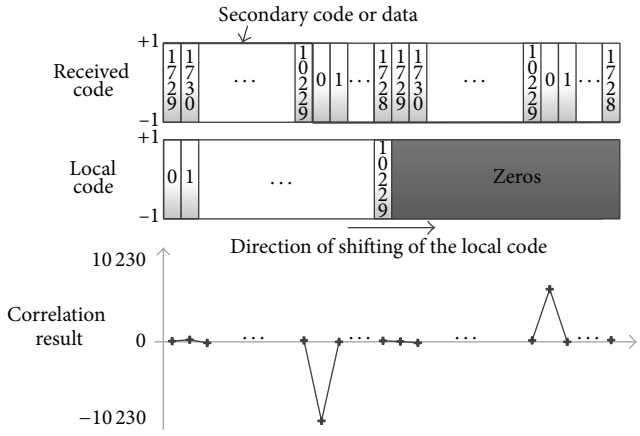

Figure 3: Straightforward solution to the bit sign transition problem. The magnitude of the first peak is always maximum, whereas the second peak can be reduced because of the sign transitions.

frequency is in the lower part of these ranges, this solution is not computationally efficient because it implies a lot of zero-padding and discarded samples.

Note that the sampling frequencies given here are those required for the acquisition. These frequencies can be the ones used by the RF front-end, but it is also possible to have an RF front-end using a higher sampling frequency and to have a resampling performed before the acquisition block (see [13], Chapter 10).

2.2. Other Solutions in the Literature. In the literature, there are different propositions to overcome this problem other than the straightforward solution. For example, some authors proposed performing averaging in the Doppler search space [8]; having two steps to first find the code delay and then the Doppler frequency [6]; or generating two local codes, with and without a transition, requiring the computation of five FFTs instead of the usual three [1, 5]. However, since these methods change the essential correlation operation (the result is not only a usual correlation between the incoming and local signals), there is a negative impact on the signal-to-noise ratio (SNR) and thus on the probability of detection compared to the 3-FFT solution. Another approach was developed in [3], where it is proposed to perform bit synchronization collectively using several satellites.

In a recent paper [4], the authors proposed an algorithm that reduces the complexity by computing less samples that are further discarded, while not affecting the SNR (because the required correlation samples are computed accurately). With this proposed algorithm, the zero-padding could be reduced, and it was shown that it was possible to compute five FFTs of 32 768 points instead of three FFTs of 65 536 points. Later, a slightly different algorithm was found (still computing five FFTs of 32 768 points but removing the need of a memory to store an intermediate result), using slightly less memory resources for an FPGA-based receiver (see [17, Chap. 5]). Next, this solution will be called the 5-FFT solution (as for the 3-FFT solution, the FFT of the local code could be precomputed, but we do not consider this case here). Note that this solution has a complexity similar to the algorithms proposed in [1, 5], since they all compute five FFTs of same length. The difference is that our algorithm computes exactly the correlation and thus the detection performance is not affected but the algorithm can be applied only for a certain range of sampling frequencies, whereas the algorithms in [1, 5] affect the detection performance but they can be applied for any sampling frequency; therefore, the best solution depends on the context.

In summary, with the 5-FFT solution proposed in [17, Chap. 5], the number of operations is reduced by about 20% compared to the 3-FFT solution, which is potentially interesting for digital signal processor based receivers; and the required memory is reduced by about 50% for the same processing time for FPGA-based receivers. However, this algorithm can be used only for sampling frequencies between 20.46 and 21.846 MHz for the L5, E5a, and E5b signals and at most of 5.4615 MHz for the E1 signal (this low sampling frequency implies additional losses due to the higher code step in the acquisition) [17, Chap. 5].

In the next section, a different approach is presented, still with the same goal, namely, computing exactly the correlation and reducing the complexity by decreasing the amount of zero-padding.

3. New Solution Based on Smaller Size FFTs

It is known that an FFT can be computed by combining the results of several smaller FFTs, at the expense of an increase in the number of operations (see [13], Section 13.47). This can be useful for our problem because the zero-padding can be reduced, as shown in this section.

3.1. Computing an FFT Using Three Smaller FFTs. The discrete Fourier transform (DFT) of a sequence x_n of N points is defined as

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}, \tag{1}$$

with $k = 0, 1, \ldots, N-1$. If we divide the input sequence x_n into three consecutive sections of $N/3$ points (assuming that N is a multiple of three), we have

$$\begin{aligned} X_k = &\sum_{n=0}^{N/3-1} x_n e^{-j2\pi kn/N} + \sum_{n=N/3}^{2N/3-1} x_n e^{-j2\pi kn/N} \\ &+ \sum_{n=2N/3}^{N-1} x_n e^{-j2\pi kn/N}. \end{aligned} \tag{2}$$

Then, changing the index n to $n_2 = n - N/3$ and $n_3 = n - 2N/3$ in the second and third sum, respectively, we obtain

$$\begin{aligned} X_k &= \sum_{n=0}^{N/3-1} x_n e^{-j2\pi kn/N} + \sum_{n_2=0}^{N/3-1} x_{n_2+N/3} e^{-j2\pi k(n_2+N/3)/N} \\ &\quad + \sum_{n_3=0}^{N/3-1} x_{n_3+2N/3} e^{-j2\pi k(n_3+2N/3)/N} \\ &= \sum_{n=0}^{N/3-1} x_n e^{-j2\pi kn/N} \\ &\quad + \sum_{n_2=0}^{N/3-1} x_{n_2+N/3} e^{-j2\pi k/3} e^{-j2\pi kn_2/N} \\ &\quad + \sum_{n_3=0}^{N/3-1} x_{n_3+2N/3} e^{-j4\pi k/3} e^{-j2\pi kn_3/N}. \end{aligned} \tag{3}$$

Finally, we can group again the terms in the same sum to obtain

$$\begin{aligned} X_k = &\sum_{n=0}^{N/3-1} \left(x_n + x_{n+N/3} e^{-j2\pi k/3} + x_{n+2N/3} e^{-j4\pi k/3} \right) \\ &\cdot e^{-j2\pi kn/N}. \end{aligned} \tag{4}$$

Then, if we divide the output sequence X_k into three sequences of $N/3$ points with a downsampling by a factor three (as done in polyphase decomposition), we obtain

$$\begin{aligned} X_{3k} &= \sum_{n=0}^{N/3-1} \left(x_n + x_{n+N/3} e^{-j2\pi(3k)/3} + x_{n+2N/3} e^{-j4\pi(3k)/3} \right) e^{-j2\pi(3k)n/N} \\ &= \sum_{n=0}^{N/3-1} \underbrace{\left(x_n + x_{n+N/3} + x_{n+2N/3} \right)}_{\triangleq x_{0,n}} e^{-j2\pi kn/(N/3)} \\ &= \sum_{n=0}^{N/3-1} x_{0,n} e^{-j2\pi kn/(N/3)}, \\ X_{3k+1} &= \sum_{n=0}^{N/3-1} \left(x_n + x_{n+N/3} e^{-j2\pi(3k+1)/3} + x_{n+2N/3} e^{-j4\pi(3k+1)/3} \right) e^{-j2\pi(3k+1)n/N} \\ &= \sum_{n=0}^{N/3-1} \underbrace{\left(x_n + x_{n+N/3} e^{-j2\pi/3} + x_{n+2N/3} e^{j2\pi/3} \right)}_{\triangleq x_{1,n}} e^{-j2\pi n/N} e^{-j2\pi kn/(N/3)} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{n=0}^{N/3-1} x_{1,n} e^{-j2\pi n/N} e^{-j2\pi kn/(N/3)}, \\
X_{3k+2} &= \sum_{n=0}^{N/3-1} \left(x_n + x_{n+N/3} e^{-j2\pi(3k+2)/3} + x_{n+2N/3} e^{-j4\pi(3k+2)/3} \right) e^{-j2\pi(3k+2)n/N} \\
&= \sum_{n=0}^{N/3-1} \underbrace{\left(x_n + x_{n+N/3} e^{j2\pi/3} + x_{n+2N/3} e^{-j2\pi/3} \right)}_{\triangleq x_{2,n}} e^{-j4\pi n/N} e^{-j2\pi kn/(N/3)} \\
&= \sum_{n=0}^{N/3-1} x_{2,n} e^{-j4\pi n/N} e^{-j2\pi kn/(N/3)},
\end{aligned}
\tag{5}
$$

with $k = 0, 1, \ldots, N/3 - 1$. Therefore an N-point DFT (or FFT) can be computed using three $N/3$-point DFTs (FFTs), as shown in Figure 4, where the combination block includes the elements to compute the operation in parentheses in (5). There are different ways to compute these operations, discussed in Appendix A. The same development can be done for an inverse DFT, providing a symmetric schematic.

A similar development can be done for the computation of an N-point FFT using five $N/5$-point FFTs (assuming that N is a multiple of five). The computation of the combinations for this case is discussed in Appendix B.

3.2. Application to the GNSS Signal Acquisition Problem. Using the developments of the previous section to compute one FFT using three FFTs, it is thus possible to compute a 49 152-point FFT using three 16 384-point FFTs. Thus, using this approach for the acquisition of the L5, E5a, E5b, and E1 signals, the sequences would need to be zero-padded only up to 49 152 (much less than 65 536). The reduction of the zero-padding directly reduces the complexity of the acquisition, as shown in the next section, with great benefits for its implementation. Next, this solution will be called the 9-FFT solution.

This method works if the initial length of the sequence is lower than or equal to 49 152, which corresponds to a maximum sampling frequency of 24.576 MHz for the L5, E5a, and E5b signals and to 6.144 MHz for the E1 signal. Therefore, the range of applicability of this solution is larger than that of the 5-FFT solution (21.846 MHz for the L5, E5a, and E5b signals, 5.4615 MHz for the E1 signal [4], [17, Chap. 5]), and the proposed algorithm does not introduce any loss with the E1 signal.

Similarly, it would be possible to compute a 40 960-point FFT using five 8192-point FFTs. The sequences would need to be zero-padded only up to 40 960; that is, only 20 zeros would be padded. However, this method works if the initial length of the sequence is lower than or equal to 40 960, which corresponds to a maximum sampling frequency of 20.48 MHz for the L5, E5a, and E5b signals and to 5.120 MHz for the E1 signal. Therefore, the range of applicability in this case is very small, and it implies a loss for the E1 signal. Next, this solution will be called the 15-FFT solution.

Remember that the sampling frequencies given are at the acquisition stage. The actual sampling frequency used by the RF front-end can be higher if a resampling is performed before the acquisition.

It is clear that it is not interesting to use higher splitting in our case. Indeed, an FFT can also be split into seven smaller FFTs; however, with seven 4096-point FFTs we can compute a 28 672-point FFT, which is much below the length of our signal (40 960 or 49 104); and with seven 8192-point FFTs we can compute a 57344-point FFT, which is much higher than the length of our signal; therefore, such a solution would be less efficient (because there would be more zero-padding and because the combinations of the FFTs inputs would become much more complex), and the only advantage is that such solution could be used for higher sampling frequency, for example, up to 28.672 MHz for the L5, E5a, and E5b signals.

4. Comparison of the Solutions

In this section, we compare the 3-FFT solution, the 5-FFT solution (proposed in [17, Chap. 5]), and the two solutions proposed in this paper: the 9-FFT and 15-FFT solutions. The schematics of these solutions are given in Figure 5, except for the 15-FFT solution, which can be easily obtained looking at the 9-FFT implementation. As mentioned previously, for each solution, the FFTs of the local code could be precomputed and stored in a memory to reduce the actual number of FFTs; however, in this paper we do not consider these cases, although the reader can transpose the results easily to these cases.

First, we review the sampling frequency range applicable to the different solutions. Then, we compare the complexity by evaluating the number of operations, which can be interesting for a digital signal processor based receiver. Finally, we compare the resources for an FPGA-based receiver. Remember that the 5-FFT solution has a complexity similar to the algorithms proposed in [1, 5] (since they all compute five FFTs of same length); thus, the following results regarding

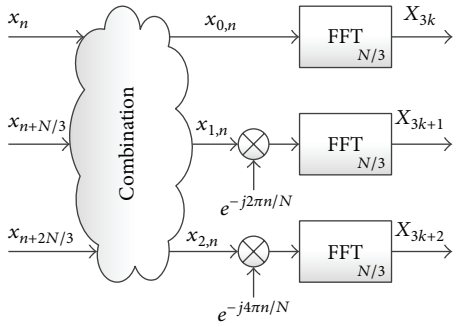

Figure 4: Computation of an FFT of N points using three FFTs of $N/3$ points.

the complexity and the FPGA implementations for the 5-FFT solution are valid for these algorithms too.

4.1. Sampling Frequency. As mentioned in the previous sections, each algorithm can be applied for a certain range of sampling frequency. These ranges are summarized in Table 1.

Note however that these ranges are for an exact computation of the code correlation, and it is possible to use a higher sampling frequency at the cost of a potential loss. For example, it would be possible to use a sampling frequency of 26 MHz with the 9-FFT solution. In this case, two code periods would correspond to 52 000 samples, and by removing the last 2848 samples the circular correlation can be performed on the 49 152 remaining samples. This can lead to a little loss, depending on the received code delay. If the delay of the code is between 0 and 23 152 (49 152–26 000), there will be no loss. If the delay is higher, a portion of the local code will not be aligned with the received code, introducing a loss. In the worst case of this example, 2848 samples will not be aligned, leading to a loss of about 1 dB. Of course, the higher the sampling frequency is, the higher the potential loss becomes.

4.2. Complexity Analysis. To evaluate the implementations' complexity, the number of complex multiplications and additions is computed. For this, we consider that an N-point FFT requires $N/2\log_2(N)$ complex multiplications and $N\log_2(N)$ complex additions [13].

The summary of the complexity analysis is given in Table 2. For the 9-FFT solution, one complex multiplier and seven complex adders are considered for each combination (see Appendix A), and, for the 15-FFT solution, four complex multipliers and 24 complex adders are considered for each combination (see Appendix B).

Therefore, for a digital signal processor based receiver, the 9-FFT solution is not significantly better than the 5-FFT solution, but the 15-FFT is (20% less multiplications and 11% less additions).

4.3. FPGA Implementation and Analysis. To compare the implementations on an FPGA, we compare the resources in terms of logic elements, memory size, and number of multipliers, when the implementations provide approximately the same processing time. For this evaluation, we consider the Altera Stratix V FPGA family [23] and the FFT core provided by Altera [24], with the streaming data flow and 16 bits of resolution for the input, output, and twiddle factor [24].

The resources of the different elements of the implementations have been evaluated separately. The FFTs resources have been estimated after place and route of a design containing one FFT with the Quartus II software 14.0, the adders and multipliers are estimated using models given in [20] (adapting the models from the Stratix III to the Stratix V FPGA family), and the control signals have been neglected. Note that this analysis is an estimation, which can slightly fluctuate depending on the FPGA family, on the complete design inside the FPGA, and on the FFT parameters.

For the 3-FFT solution, we consider the implementation given by Figure 5(a). For the other solutions, the direct FPGA implementation of Figures 5(b) and 5(c) would lead to an increase of the resources and to a decrease of the processing time by factors 2 and 4, respectively (which is the ratio in the FFTs length) [17, Chap. 4 and 5]. Therefore, we will consider time multiplexing for the implementations to use fewer FFTs and to obtain approximately the same processing time as the 3-FFT FPGA implementation of Figure 5(a).

For the 5-FFT solution, the time-multiplexing implementation is given in Figure 6, where $y_{M0,n}$ is computed first and $y_{M1,n}$ is computed next. Since the FFTs length is half the FFT length of the 3-FFT solution and that the FFTs are used twice to get a complete correlation result, the processing time is the same as with Figure 5(a) (see [17, Chap. 4 and 5] for more details).

For the 9-FFT solution, the time-multiplexing implementation is given in Figure 7. First, the three sections of h_n and x_n are accessed in parallel and combined to obtain $h_{0,n}$ and $x_{0,n}$, respectively; their circular correlation is computed with the FFTs; and the result $y_{0,n}$ is stored in a memory. Then, this is repeated with $h_{1,n}$ and $x_{1,n}$, which are multiplied by a complex exponential before the FFTs; the IFFT output is also multiplied by a complex exponential to obtain $y_{1,n}$, also stored in memory. Finally, this operation is repeated with $h_{2,n}$ and $x_{2,n}$; when $y_{2,n}$ is available, the memories are read, and $y_{0,n}$, $y_{1,n}$, and $y_{2,n}$ are combined to obtain the three sections of the correlation output. With this implementation, since the FFTs length is divided by four compared to the 3-FFT solution and the FFTs are used three times to get a complete correlation result, the processing time is 75% the one of Figures 5(a) and 6.

The same approach is considered for the 15-FFT solution. In this case, the FFTs length is divided by eight compared to the 3-FFT solution, and the FFTs are used five times to get a complete correlation result; therefore, the processing time is 62.5% the one of Figures 5 and 6.

The resources used by each implementation are given in Table 3, in terms of adaptive logic module (ALM, element containing a bit of logic and four registers), memory blocks of 20 Kibit (i.e., of 20 × 1024 bits, denoted M20K), and digital signal processing (DSP) blocks (containing two 18-bit hardware multipliers); see [23] for more details about the elements inside Stratix V FPGAs. The details of the resource's estimation can be found in Table 4.

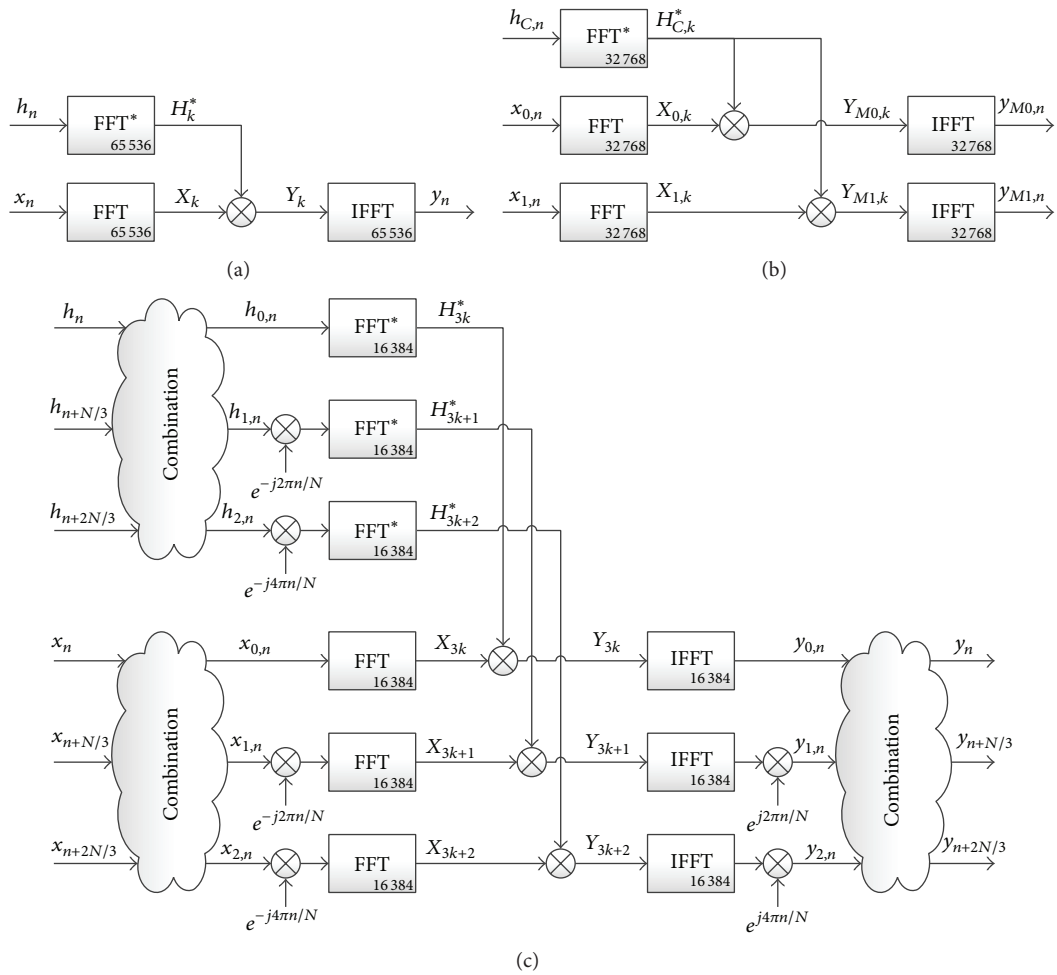

Figure 5: Schematic of the different solutions: (a) 3-FFT solution (straightforward solution), (b) 5-FFT solution (proposed in [17, Chap. 5]), and (c) 9-FFT solution (proposed in this paper).

Table 1: Range of applicability of the different solutions.

Solution	FFT length	Sampling frequency range for L5, E5a, and E5b signals	Sampling frequency range for the E1 signal
3-FFT solution	65 536	20.46 MHz–32.768 MHz	6.138 MHz–8.192 MHz
5-FFT solution	32 768	20.46 MHz–21.846 MHz	5.4615 MHz*
9-FFT solution	16 384	20.46 MHz–24.576 MHz	6.138 MHz–6.144 MHz
15-FFT solution	8192	20.46 MHz–20.480 MHz	5.120 MHz*

A $*$ indicates that there is a loss due to a higher code step during the acquisition.

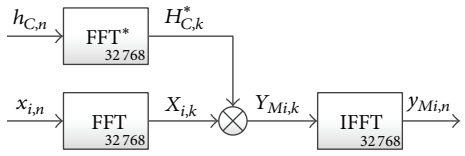

Figure 6: FPGA implementation of the solution proposed in [17], Chapter 5. $i \in 0,1$.

In Table 3, which summarizes the resources, it can be verified that the implementation of the 5-FFT solution uses less than half the memory of the 3-FFT solution (the reduction is 56.6% whereas it was about 50% in [17] because of the different FPGA families considered) and slightly less logic.

With the proposed 9-FFT solution, the memory is again reduced, by 39.8% compared to the 5-FFT solution; however, the logic is slightly higher (+6.4%, in part because the 16 384-point FFT requires more logic than the 32 768-point FFT), and the DSP blocks is also higher (+21.1%, because of the multiplication with the complex exponentials and the product between the FFTs outputs). However, the memory saving is more valuable than the increase of the DSP blocks

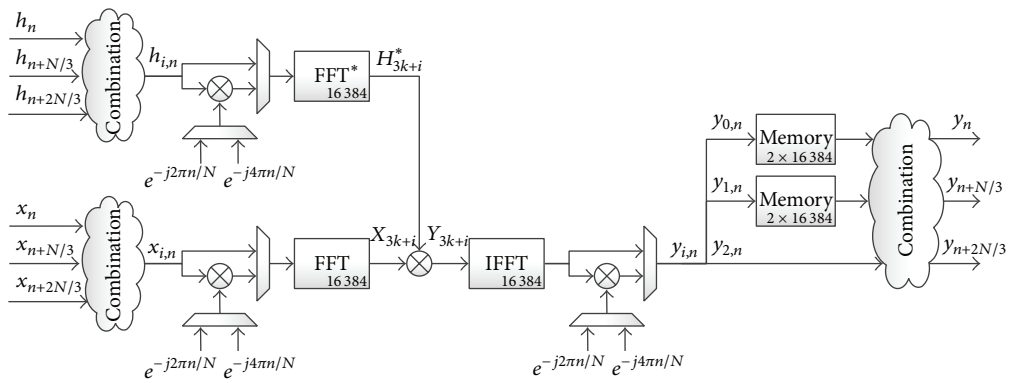

Figure 7: FPGA implementation of the proposed solution.

(except in the cases where the overall system implemented into the FPGA strongly limits the number of DSP blocks available). Indeed, the minimum number of DSP blocks in a Stratix V FPGA is 256 [23]; therefore, the 8 additional DSP blocks represent 3.13% of the total available. For the same FPGA, the memory can be between 957 and 2560 M20K blocks. Therefore, the 308 M20K blocks saved (equivalent to about 6 Mbit) represent between 32.2% and 12.0% of the total available. As for the logic, the 531 additional ALMs represent less than 1% of the total available (minimum of 128 300 ALMs). Thus, it can be concluded that the gain of memory is much more valuable than the loss of other elements.

With the proposed 15-FFT solution, the memory is further reduced, by 63.9% compared to the 5-FFT solution, but the number of DSP blocks is again increased. However, the number of additional DSP blocks represents at most 5.5% of the total available in a Stratix V FPGA, whereas the number of M20K blocks saved represents between 20% and 53% of the total available.

It can be noted that, compared to the 3-FFT algorithm, the memory used by the 9-FFT and 15-FFT algorithms is, respectively, 3.8 and 6.4 times smaller, which represents a huge amount of memory saved, and allows the implementation of the algorithm in the smallest of the Stratix V FPGAs.

5. Conclusions

This paper discussed the problem of the complexity of a circular correlation computed by FFTs when two code periods are used. This problem appears in the parallel code search acquisition of GNSS signals, when we want to manage the sign transitions due to data or secondary code or when we need to zero-pad the signals to reach a specific FFT length (e.g., a power of two).

With the structure of new GPS L5 and Galileo E5a, E5b, and E1 BOC(1,1) signals, the straightforward solution to this problem uses three 65 536-point FFTs (3-FFT solution) for sampling frequencies between 20.46 and 32.768 MHz for the L5, E5a, and E5b signals and between 6.138 and 8.192 MHz for the E1 signal processed as BOC(1,1). However, for sampling frequencies in the lower part of these ranges, this solution is not computationally efficient because it implies a lot of zero-padding.

Table 2: Number of operations of the different solutions.

Solution	Number of complex multiplications	Number of complex additions
3-FFT solution	1 638 400	3 145 728
5-FFT solution	1 294 336 (−21%)	2 457 600 (−21.9%)
9-FFT solution	1 228 800 (−25%)	2 408 448 (−23.4%)
15-FFT solution	1 036 288 (−36.8%)	2 187 264 (−30.5%)

The percentage is the reduction compared to the 3-FFT solution.

Previously, in [4, 17], we proposed an alternative algorithm that computes exactly the output samples (i.e., without any approximation), but using five 32 768-point FFTs (5-FFT solution). The main benefits were a reduction of the number of operations of about 20% and a reduction of the memory resources of about 50% for an FPGA implementation compared to the 3-FFT solution, without increasing other resources (logic and multipliers) nor the processing time. However, the 5-FFT solution can be used only for sampling frequencies between 20.46 and 21.846 MHz for the L5, E5a, and E5b signals and at most of 5.4615 MHz for the E1 signal (this low sampling frequency implying additional losses due to the higher code step in the acquisition).

In this paper, we proposed two algorithms that still compute exactly the output samples, one using nine 16 384-point FFTs (9-FFT solution) and one using fifteen 8192-point FFTs (15-FFT solution), respectively. These algorithms exploit the fact that an FFT can be computed using K FFTs of length K times smaller. Therefore, using three 16 384-point FFTs, it is possible to compute a 49 152-point FFT and thus to reduce the zero-padding necessary compared to the previous algorithms. Likewise, with five 8192-point FFTs, it is possible to compute a 40 960-point FFT.

Compared to the 5-FFT solution, the 9-FFT solution has a slightly lower number of operations, and for an FPGA

Table 3: Comparison of the resources for the different algorithms using the Altera FFT for the L5, E5a, E5b, and E1 signals, considering time multiplexing.

Implementation	Logic usage (ALM)	Memory usage (M20K)	Multipliers usage (DSP blocks)
3-FFT solution (Figure 5(a))			
Total	8760	1824	38
5-FFT solution (Figure 6)			
Total	8274	792	38
Difference with 3-FFT solution	−486 (−5.5%)	−1032 (−56.6%)	0
Proposed 9-FFT solution (Figure 7)			
Total	8805	484	46
Difference with 3-FFT solution	+45 (+0.5%)	−1340 (−73.5%)	+8 (+21.1%)
Difference with 5-FFT solution	+531 (+6.4%)	−308 (−38.9%)	+8 (+21.1%)
Proposed 15-FFT solution			
Total	8645	286	52
Difference with 3-FFT solution	−115 (−1.3%)	−1538 (−84.3%)	+14 (+36.8%)
Difference with 5-FFT solution	+371 (+4.5%)	−506 (−63.9%)	+14 (+36.8%)

Table 4: Details of the resources for the different algorithms using the Altera FFT for the L5, E5a, E5b, and E1 signals, considering time multiplexing.

Implementation	Function	Logic usage (ALM)	Memory usage (M20K)	Multipliers usage (DSP blocks)
3-FFT solution (Figure 5(a))	3 FFTs (65 536 points)	3 × 2920	3 × 608	3 × 12
	1 multiplier	0	0	2
	Total	8760	1824	38
5-FFT solution (Figure 6)	3 FFTs (32 768 points)	3 × 2758	3 × 264	3 × 12
	1 multiplier	0	0	2
	Total	8274	792	38
Proposed 9-FFT solution (Figure 7)	3 FFTs (16 384 points)	3 × 2883	3 × 140	3 × 12
	4 multipliers	0	0	4 × 2
	Combinations for h_n	28	0	0
	Combinations for x_n	64	0	1
	Combinations for y_n	64	0	1
	2 memories (32 768 points)	0	2 × 32	0
	Total	8805	484	46
Proposed 15-FFT solution	3 FFTs (8192 points)	3 × 2695	3 × 74	3 × 12
	4 multipliers	0	0	4 × 2
	Combinations for h_n	112	0	0
	Combinations for x_n	224	0	4
	Combinations for y_n	224	0	4
	4 memories (16 384 points)	0	4 × 16	0
	Total	8645	286	52

implementation the processing time is reduced by 25%, and the memory is reduced by about 40% (which means a reduction of about 75% compared to the straightforward algorithm). In return, there is a small increase of the logic and DSP blocks (+6% and +21%, resp.) but, compared to the resources available in an FPGA, the memory saved represents more than the DSP blocks lost. Moreover, this algorithm can be used for sampling frequencies between 20.46 and 24.576 MHz for the L5, E5a, and E5b signals and between 6.138 and 6.144 MHz for the E1 signal, that is, larger ranges compared to the 5-FFT solution. Therefore, this algorithm is overall more efficient and more versatile than the 5-FFT solution.

Regarding the 15-FFT algorithm, compared to the 5-FFT solution, the number of operations is reduced (20% less multiplications and 11% less additions), which is interesting for software defined DSP based receivers. For an FPGA implementation, the processing time is reduced by 37.5% and the memory is reduced by about 64%, for an increase of the logic and DSP blocks, but again this increase is small compared to the resource available in an FPGA. However, the sampling frequency range is more limited since it should be between 20.46 and 20.48 MHz for the L5, E5a, and E5b signals and at most of 5.12 MHz for the E1 signal, implying additional losses due to the higher code step in the acquisition.

In conclusion, two algorithms were proposed that reduce significantly the processing time and the memory resources compared to previously proposed algorithms, one providing better performance than the other but for a limited range of sampling frequency. Therefore, the context will indicate which one is the most interesting. Note also that the mentioned sampling frequencies are those at the acquisition stage; the actual sampling frequency used by the RF front-end can be higher if a sample rate conversion is performed before the acquisition.

Appendices

A. Combinations of the FFT Inputs for the 9-FFT Algorithm

When computing an FFT of N points using three FFTs of $N/3$ points, we need to perform the following combinations between the sections of the input:

$$\begin{aligned} x_{0,n} &= x_n + x_{n+N/3} + x_{n+2N/3}, \\ x_{1,n} &= x_n + x_{n+N/3}e^{-j2\pi/3} + x_{n+2N/3}e^{j2\pi/3}, \\ x_{2,n} &= x_n + x_{n+N/3}e^{j2\pi/3} + x_{n+2N/3}e^{-j2\pi/3} \end{aligned} \tag{A.1}$$

with $n = 0, 1, \ldots, N/3 - 1$. Using a matrix notation, this gives

$$\begin{bmatrix} x_{0,n} \\ x_{1,n} \\ x_{2,n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & e^{-j2\pi/3} & e^{j2\pi/3} \\ 1 & e^{j2\pi/3} & e^{-j2\pi/3} \end{bmatrix} \begin{bmatrix} x_n \\ x_{n+N/3} \\ x_{n+2N/3} \end{bmatrix}. \tag{A.2}$$

This section details different ways to compute these combinations and discusses their FPGA implementation.

A.1. Methods to Reduce the Number of Multipliers. The direct computation of these combinations implies 4 complex multipliers and 6 complex adders, that is, 16 real multipliers and 20 real adders (assuming that a complex multiplication requires 4 real multiplications and 2 additions).

However, it is possible to exploit the fact that $x_{n+N/3}$ and $x_{n+2N/3}$ are multiplied by a complex number and its conjugate. Indeed, denoting $x_{r,n}$ and $x_{i,n}$, the real and imaginary part of x_n, respectively, we have

$$\begin{aligned} x_{n+N/3}e^{-j2\pi/3} &= \left(x_{r,n+N/3} + jx_{i,n+N/3}\right)\left(-\frac{1}{2} - j\frac{\sqrt{3}}{2}\right) \\ &= \left(-\frac{x_{r,n+N/3}}{2} + \frac{\sqrt{3}x_{i,n+N/3}}{2}\right) \\ &\quad + j\left(-\frac{\sqrt{3}x_{r,n+N/3}}{2} - \frac{x_{i,n+N/3}}{2}\right), \\ x_{n+N/3}e^{j2\pi/3} &= \left(x_{r,n+N/3} + jx_{i,n+N/3}\right)\left(-\frac{1}{2} + j\frac{\sqrt{3}}{2}\right) \\ &= \left(-\frac{x_{r,n+N/3}}{2} - \frac{\sqrt{3}x_{i,n+N/3}}{2}\right) \\ &\quad + j\left(\frac{\sqrt{3}x_{r,n+N/3}}{2} - \frac{x_{i,n+N/3}}{2}\right). \end{aligned} \tag{A.3}$$

Therefore, these two complex products can be computed using only 4 real multipliers (counting the multiplication by $-1/2$) and 4 real adders. Thus (A.1) can be computed with 8 real multipliers and 20 real adders.

There is a third option to reduce the complexity, by exploiting the fact that $x_{n+N/3}$ and $x_{n+2N/3}$ are both multiplied by the same complex number and its conjugate. It is well known that the following operation

$$\begin{bmatrix} a & b \\ b & a \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} ay + bz \\ by + az \end{bmatrix} \tag{A.4}$$

can be computed as

$$\begin{aligned} &\frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} (a+b)(y+z) \\ (a-b)(y-z) \end{bmatrix} \\ &\quad = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} ay + bz + by + az \\ ay + bz - by - az \end{bmatrix}, \end{aligned} \tag{A.5}$$

requiring 2 multiplications and 6 additions (not counting the factor 1/2) instead of 4 multiplications and 2 additions. Applying this to our combination, we have

$$\begin{aligned} &\begin{bmatrix} e^{-j2\pi/3} & e^{j2\pi/3} \\ e^{j2\pi/3} & e^{-j2\pi/3} \end{bmatrix}\begin{bmatrix} x_{n+N/3} \\ x_{n+2N/3} \end{bmatrix} \\ &\quad = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} -\frac{1}{2} & 0 \\ 0 & -j\frac{\sqrt{3}}{2} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x_{n+N/3} \\ x_{n+2N/3} \end{bmatrix} \end{aligned} \tag{A.6}$$

which requires only 4 real multiplications (counting the multiplication by $-1/2$) and 8 real additions. Thus, (A.1) can be computed with 4 real multipliers and 16 real adders.

A.2. FPGA Implementation. First, implementing (A.6) in an FPGA requires only 2 real multipliers instead of 4, because the multiplication by $1/2$ can be performed by a right shift of the bits of the signal. Thus, the combinations of the sections of x_n can be implemented with two real multipliers and 16 real adders.

For the local code replica h_n, two properties can be used to further reduce the resources; it is binary and real. Since h_n is binary, the first sum and difference in (A.6) can be neglected, and the multiplication by $\sqrt{3}/2$ can be replaced by a simple change of sign according to the value of the sample (which is equivalent to an adder in terms of resources). Since h_n is real, only 3 real adders are thus needed to compute (A.6). Therefore, the combinations of the sections of h_n can be implemented with 7 real adders only. Note that this is a basic evaluation; for example, the adders used to compute $h_{0,n}$ have a much lower resolution than the others since the signals added together are binary, so they will not consume the same amount of resources. Anyway, these adders represent a very little percentage of the total resources used as shown in Table 4.

It is also possible to exploit more the characteristics of the signals, although this would not allow a significant reduction of the resources. Indeed, if the second half of h_n contains only zeros, $h_{n+2N/3}$ contains only zeros, and therefore the first sum and difference in (A.6) (neglected in the previous paragraph) are not needed, and the combinations could be simplified by removing one adder for the computation of $h_{0,n}$. Then, if we do not care about the second half of the output y_n, it is not necessary to compute $y_{n+2N/3}$, which would remove four adders.

B. Combinations of the FFT Inputs for the 15-FFT Algorithm

When computing an FFT of N points using five FFTs of $N/5$ points, we need to perform the following combinations between the sections of the input:

$$\begin{bmatrix} x_{0,n} \\ x_{1,n} \\ x_{2,n} \\ x_{3,n} \\ x_{4,n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-j2\pi/5} & e^{-j4\pi/5} & e^{j4\pi/5} & e^{j2\pi/5} \\ 1 & e^{-j4\pi/5} & e^{j2\pi/5} & e^{-j2\pi/5} & e^{j4\pi/5} \\ 1 & e^{j4\pi/5} & e^{-j2\pi/5} & e^{j2\pi/5} & e^{-j4\pi/5} \\ 1 & e^{j2\pi/5} & e^{j4\pi/5} & e^{-j4\pi/5} & e^{-j2\pi/5} \end{bmatrix} \begin{bmatrix} x_n \\ x_{n+N/5} \\ x_{n+2N/5} \\ x_{n+3N/5} \\ x_{n+4N/5} \end{bmatrix}, \tag{B.1}$$

with $n = 0, 1, \ldots, N/5 - 1$. The direct computation of this equation implies 16 complex multipliers and 20 complex adders, that is, 64 real multipliers and 72 real adders. However, rewriting the equation as follows

$$\begin{bmatrix} x_{0,n} \\ x_{1,n} \\ x_{4,n} \\ x_{2,n} \\ x_{3,n} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & e^{-j2\pi/5} & e^{j2\pi/5} & e^{-j4\pi/5} & e^{j4\pi/5} \\ 1 & e^{j2\pi/5} & e^{-j2\pi/5} & e^{j4\pi/5} & e^{-j4\pi/5} \\ 1 & e^{-j4\pi/5} & e^{j4\pi/5} & e^{j2\pi/5} & e^{-j2\pi/5} \\ 1 & e^{j4\pi/5} & e^{-j4\pi/5} & e^{-j2\pi/5} & e^{j2\pi/5} \end{bmatrix} \begin{bmatrix} x_n \\ x_{n+N/5} \\ x_{n+4N/5} \\ x_{n+2N/5} \\ x_{n+3N/5} \end{bmatrix}, \tag{B.2}$$

we find again the pattern present in (A.2). Applying the third option described in Section A.1, (B.2) can be computed with 16 real multipliers and 52 real adders.

Regarding the FPGA implementation, now neither the real nor the imaginary part of the complex exponentials is a rational number. Therefore, 16 real multipliers and 52 real adders are needed for the combinations of sections x_n. The same amount is needed for the combinations to get the sections of y_n (although 8 real adders could be saved since the last two sections of y_n are not wanted).

For h_n, since it is binary and real, only 4 real adders are needed to compute the operation similar as (A.6). Therefore, the combinations of the sections of h_n can be implemented with 28 real adders only.

C. Details of the FPGA Resources

The details of the FPGA resources for the different solutions are given in Table 4.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

The authors thank Marc-Antoine Fortin for its suggestions that improved this paper.

References

[1] C. Zhu and X. Fan, "Weak global navigation satellite system signal acquisition with bit transition," *IET Radar, Sonar and Navigation*, vol. 9, no. 1, pp. 38–47, 2015.

[2] M. Foucras, B. Ekambi, F. Bacard, O. Julien, and C. Macabiau, "Optimal GNSS acquisition parameters when considering bit transitions," in *Proceedings of the IEEE/ION Position, Location*

and Navigation Symposium (PLANS '14), pp. 804–817, IEEE, Monterey, Calif, USA, May 2014.

[3] T. Ren and M. Petovello, "Collective bit synchronization for weak GNSS signals using multiple satellites," in *Proceedings of the IEEE/ION Position, Location and Navigation Symposium (PLANS '14)*, pp. 547–555, Monterey, Calif, USA, May 2014.

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

[5] S. Jeon, H. So, G. Kim, C. Kee, K. Kwon, and S. W. Choi, "Analysis of GNSS signal acquisition methods for the bit-transition problem for a single code period," *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 56, no. 1, pp. 31–41, 2013.

[6] K. Sun and L. Lo Presti, "Bit sign transition cancellation method for GNSS signal acquisition," *Journal of Navigation*, vol. 65, no. 1, pp. 73–97, 2012.

[7] 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, 2011.

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

[9] D. Borio, M. Fantino, and L. Lo Presti, "The impact of the Galileo signal in space in the acquisition system," in *Satellite Communications and Navigation Systems*, E. Del Re and M. Ruggieri, Eds., Signals and Communication Technology, pp. 151–167, Springer, New York, NY, USA, 2008.

[10] 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 *Proceedings of the IEEE/ION Position, Location and Navigation Symposium (PLANS '10)*, pp. 745–751, Indian Wells, Calif, USA, May 2010.

[11] N. I. Ziedan, "Acquisition and fine acquisition of weak GPS L2C and L5 signals under high dynamic conditions for limited-resource applications," in *Proceedings of the 18th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS '05)*, pp. 1577–1588, September 2005.

[12] 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 '12)*, pp. 365–383, Nashville, Tenn, USA, September 2012.

[13] R. G. Lyons, *Understanding Digital Signal Processing*, Prentice-Hall, 3rd edition, 2010.

[14] C. Yu and M.-H. Yen, "Area-efficient 128- to 2048/1536-point pipeline FFT processor for LTE and mobile WiMAX systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1793–1800, 2014.

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

[16] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*, Applied and Numerical Harmonic Analysis, Birkhäuser, Boston, Mass, USA, 2007.

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

[18] P. K. Sagiraju, G. V. S. Raju, and D. Akopian, "Fast acquisition implementation for high sensitivity global positioning systems receivers based on joint and reduced space search," *IET Radar, Sonar and Navigation*, vol. 2, no. 5, pp. 376–387, 2008.

[19] C.-W. Chen, S.-H. Chen, H.-W. Tsao, and W.-L. Mao, "Memory-based two-dimensional-parallel differential matched filter correlator for global navigation satellite system code acquisition," *IET Radar, Sonar and Navigation*, vol. 8, no. 5, pp. 525–535, 2014.

[20] J. Leclere, 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, 2013.

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

[22] W. De Wilde, J.-M. Sleewaegen, A. Simsky et al., "Fast signal acquisition technology for new GPS/Galileo receivers," in *Proceedings of the IEEE/ION Position, Location, and Navigation Symposium (PLANS '06)*, pp. 1074–1079, IEEE, April 2006.

[23] Altera, *Stratix V Device Handbook*, Altera, San Jose, Calif, USA, 2014.

[24] Altera, *FFT MegaCore Function User Guide*, Altera, San Jose, Calif, USA, 2014.