Comparison Framework of FPGA-based GNSS Signals Acquisition Architectures

Jérôme Leclère, Cyril Botteron, Pierre-André Farine

***Abstract*—The acquisition of Global Navigation Satellite Systems signals using Code Division Multiple Access can be performed through classical correlation or using a Fourier transform. These methods are well known but what is missing is a comparison of their performance for a given hardware area or target. This paper presents this comparison for Field-Programmable Gate Arrays, describing the different parameters involved in the acquisition, detailing some optimized implementations where hardware elements are duplicated, and estimating and discussing the performances. The influence of the Doppler effect on the code, is also discussed as it plays an important role, particularly for new signals using a high chipping rate.**

***Index Terms*—Acquisition, Architecture, CDMA, FFT, FPGA, GNSS**

I. Introduction

GLOBAL Navigation Satellite Systems (GNSSs) use pseudorandom noise (PRN) codes for ranging, and to distinguish satellites via Code Division Multiple Access (CDMA). The first processing step in a GNSS receiver is thus acquisition, which consists of the rough estimation of the phases of the PRN codes, as well as the Doppler frequencies through multiple correlations with locally generated signals (called replicas).

Acquisition can require a relatively long processing time, due to the large number of possibilities for the two parameters being estimated. This is even truer for very weak signals, which are acquired nowadays by high-sensitivity receivers, since this implies very long integration times.

Today's technology allows very efficient acquisitions, by processing signals at higher frequencies, and by parallelizing operations by duplicating hardware elements and/or using Fourier transforms. However, a general question is how to select the best architecture for a specific application.

In this context, this paper provides a general framework for hardware implementation on Field-Programmable Gate Arrays (FPGAs). The framework is illustrated with an example application in which we rank the considered architectures according to their performance. In addition, this paper also highlights and discusses the influence of the Doppler effect on the code. It will be shown that it may have a great impact on some acquisition architectures.

We review in Section II the acquisition principle before presenting the three main architectures used for the acquisition of CDMA signals with their advantages and drawbacks. These architectures are the Serial Search (SS), which is the traditional method; the Parallel Frequency Search (PFS), which uses a Fourier transform as a spectrum analyzer; and the Parallel Code-phase Search (PCS), which uses a Fourier transform to perform the correlation faster. The same section also presents the different parameters involved in acquisition and their impact on acquisition time.

In Section III, the target devices, FPGAs, are briefly presented in order to provide the tools to understand the detailed analysis of the architectures that follows. After this analysis, the analytical developments about the acquisition time are completed.

Finally, in Section IV, we examine a practical example application considering the acquisition of the Global Positioning System (GPS) L1 C/A signal, and discuss newer signals.

Note that while a GPS L1 C/A signal is considered throughout this paper for illustration purposes, the proposed framework can easily be adapted to other GNSS signals. Only the code length, frequency and type (e.g. time-multiplexed, composite, addition of sub-carrier) need to be adapted to consider other GNSS signals including those from Galileo, GLONASS, GPS, and other forthcoming GNSSs.

II. Acquisition

A. Acquisition Principle

The signal emitted by satellites is a combination of several signals : 1) a carrier; 2) a PRN code specific to each satellite used for multiplexing and ranging measurements, denoted *c(t)* (*c(t)* can also include a secondary PRN code and a subcarrier); and 3) navigation data which contain information for positioning (time, ephemeris, etc.), denoted *d(t)*. For instance, the L1 C/A signal emitted by GPS satellites can be expressed as

$$s_i(t) = a\cos(2\pi f_{L1} t)\, c_i(t)\, d_i(t) \qquad (1)$$

where a is the amplitude of the signal, f_{L1} the carrier frequency (1575.42 MHz), and i an index that denotes the satellite. The

Manuscript received June 9, 2011, revised December 19, 2011, revised May 25, 2012, revised August 23, 2012, accepted September 19, 2012.

Authors are with the Electronics and Signal Processing Laboratory (ESPLAB) of the Institute of Microengineering (IMT) of the École Polytechnique Fédérale de Lausanne (EPFL), Neuchâtel, Switzerland.

e-mail : firstname.lastname@epfl.ch

code chipping rate is 1.023 Mchip/s (a chip refers to the signal corresponding to an individual term of a pseudo random sequence [1]), and the data rate is 50 bit/s. This signal uses a BPSK modulation, whereas newer signals use more advanced modulations (e.g. QPSK, BOC, CBOC, AltBOC). The three signal components, carrier, code, and data, however, are always present, except for pilot channels, which carry no data.

At the receiver side, the signal is strongly attenuated, insomuch as the thermal noise generated by the front-end is even stronger than the signal, leading to a signal-to-noise ratio (SNR) below 0 dB. Consequently, before extracting the navigation data, the carrier and the PRN code have to be removed in order to integrate the signal over time and raise the signal out of the noise.

The removal of the carrier and the code is performed by multiplying the input signal with local replicas, i.e. a carrier at the same frequency as the input carrier, which must be complex to cancel the influence of the unknown phase of the input carrier, and a PRN code with the same phase as the input code.

The phase of the incoming PRN code is unknown on the receiver side since it depends on many quantities such as the transmit and travel times of the signal. Consequently, in order to align the replica with it, all possible phases have to be tested with a certain resolution. The resolution depends on the modulation used and on the required performance, e.g. it is typically ½ chip for the GPS L1 C/A signal.

The frequency of the received signals is affected on the one hand by the Doppler effect due to the high speed of the satellites (about ± 5 kHz) and the speed of the receiver to a lesser extent (maximum of 1.5 Hz/(km/h) for the L1 frequency [2]). On the other hand, the inaccuracy of the receiver's oscillator creates a Doppler-like effect through the down-conversion from radio frequency to baseband by placing the signal away from its nominal intermediate frequency (which is typically a low Intermediate Frequency (IF) or zero-IF), but it affects all the satellite signals in the same way. The total range of the Doppler frequency is the combination of these elements and depends on the context [2]. Consequently, in order to align the frequency of the carrier replica with the input signal, this range, called the frequency search space, has to be explored with a certain resolution. This resolution depends on the integration time used, because, after integration, a mismatch of the frequencies results in degradation as a sinc function whose width is inversely proportional to the integration time [1].

A graphical representation of the time-frequency search space is depicted in Fig. 1, where each square represents a cell, i.e. a code-phase/carrier frequency bin. The acquisition consists thus in the evaluation of a 2D correlation function, called the Ambiguity Function [3].

To increase the integration time (and thus the SNR) without affecting the resolution of the frequency search, non-coherent integrations can be performed. They consist of summing the magnitude (or the power) of many complex correlation values of the same cell. The integration performed before the computation of the magnitude (or the power) is called the coherent integration. Non-coherent integrations can also be used to extend the limit imposed by the data bits. Indeed, if there is a data bit transition during a coherent integration, it will result in a loss. More details about coherent and non-coherent integration can be found in [2, 4].

To help with the acquisition process, a receiver can get information (time, position, ephemeris, etc.) from another source (mobile network, internet, etc.). This can help the receiver 1) to know which satellites are in view; 2) to reduce the frequency search space (knowing the approximate time, and receiver and satellites positions, the corresponding Doppler frequencies can be estimated); and 3) to reduce the code-phase search space if fine time (time known to better than one code period, e.g. 1 ms for the GPS L1 C/A signal) is available.

In this paper, in addition to the stand-alone case, we consider frequency assistance. This is the most common assistance type and also the easiest to set up. It is also similar to the warm start of a receiver where the almanac and position have been memorized during the last use and the time is coarsely known.

Considering L satellites seen by the receiver, the nth sample of the received signal after in-phase and quadrature (I/Q) conversion and sampling to an IF (or zero-IF) can be written as

$$s[n] = \sum_{i=1}^{L} \left\{ a_i \exp\left[j\left(2\pi\left(f_{IF} + f_{D,i}\right)nT_S + \theta_i\right)\right] \right.$$
$$\left. c_i\left[\left(1+\frac{f_{D,i}}{f_{L1}}\right)nT_S - \tau_i\right] d_i\left(nT_S - \tau_i\right)\right\} + \eta_{IF}[n] \quad (2)$$

where f_{IF} is the intermediate frequency, $f_{D,i}$ the Doppler frequency of the ith satellite, T_S the sampling period, θ_i the phase of the carrier of the ith satellite, τ_i the phase of the PRN code of the ith satellite, and η_{IF} the noise component.

The different satellite signals can be processed in parallel through several acquisition channels or sequentially using only one acquisition channel. In this paper, we consider a system with one acquisition channel, because it is more efficient in terms of resource sharing. However, the proposed framework can be applied as well for several channels. Once a signal is acquired, it can be tracked using closed loops, the navigation data bits can be extracted, and as soon as this is performed for at least four satellites, the position of the receiver can be computed. Detailed information about GNSS signals structure and processing can be found in [5, 6].

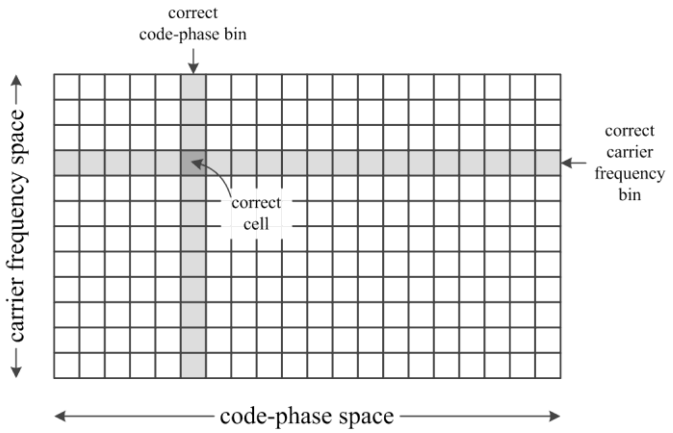

Fig. 1. Time-frequency search space

B. Acquisition Architectures

In this section, three well-known architectures widely used to perform acquisition [7] are presented with their properties, advantages, and drawbacks. In the following figures, N_C and N_{NC} are related to the coherent and non-coherent integration time, respectively (their formal definition is provided after the description of the architectures). Simple arrows are used for scalars, whereas arrows with a slash are used for vectors, with the size of the vector specified if modified by the previous operation. The operation that holds samples to form a vector is not shown in the figures (like, for example, between the coherent accumulator and the FFT in Fig. 3).

1) Serial Search

The first and oldest acquisition method is called the serial search and its block diagram is shown in Fig. 2. There are five main steps : 1) multiplication of the complex input signal ($s[n] = s_I[n] + j\, s_Q[n]$) with the local code replica; 2) multiplication with the local complex carrier replica; 3) coherent integration through an accumulator (also called integrate and dump process [6]); 4) magnitude computation; and 5) non-coherent integration of consecutive results.

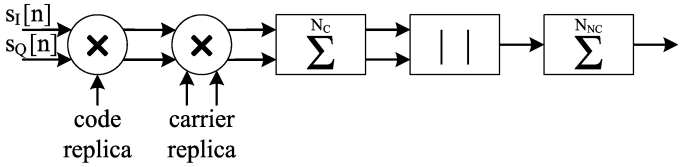

Fig. 2. Diagram of Serial Search (SS) acquisition

In this architecture, all possibilities for carrier frequency and code phase are tested sequentially. Looking at Fig. 1, this means that the cells are tested one after the other. The advantage of this architecture is its simplicity, but its drawback is the time needed to acquire the signal which is relatively long, since there are thousands of code phases to search and the number of frequency bins can be from dozens (e.g. a span of ± 5 kHz with steps of 500 Hz gives 21 bins) to hundreds (e.g. a span of ± 5 kHz with steps of 50 Hz gives 201 bins), which leads to numerous combinations.

2) Parallel Frequency Search

One solution to reduce acquisition time is to parallelize the search in the frequency space. The idea consists of performing the coherent integration on a small part of the signal (typically less than the PRN code period), and then using a Fourier transform (implemented as a Fast Fourier Transform, or FFT) on consecutive accumulation results. This allows the test of N_{FFT} frequency bins at once, which may cover the entire frequency search space [8]. After the FFT, the magnitude computation and the non-coherent integration are performed on each frequency bin, their inputs and outputs are thus vectors. This architecture is depicted in Fig. 3. In this case, the carrier replica is generated only to remove the intermediate frequency or for rough compensation of the Doppler, and the different code phases are tested sequentially. Looking at Fig. 1, this means that the columns are tested one after the other.

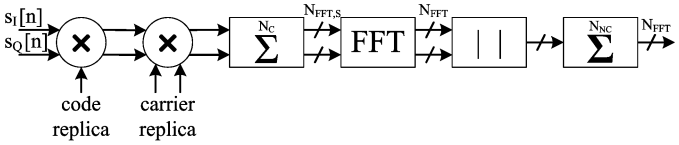

Fig. 3. Diagram of Parallel Frequency Search (PFS) acquisition

The advantage of this architecture is the large reduction in processing time, since it is equivalent to having as many correlators as there are points in the FFT (each point being equivalent to a frequency bin). In addition, the relatively small number of points in the FFT (dozens to hundreds) makes it easily implementable in many contexts.

However, this architecture has three main drawbacks. The first is a sensitivity loss for frequencies away from the center [9, 10]. Indeed, while the aim of performing the coherent accumulation before the FFT is to reduce the size of the FFT, this will also cause a loss proportional to $\mathrm{sinc}(\pi\, f_D\, N_C / f_S)$, where f_D is the Doppler frequency, N_C the number of samples used by the coherent accumulator per accumulation, and f_S the sampling frequency. The frequencies searched by the FFT are from $-f_S / (2\, N_C) + f_S / N_{FFT,S}$ to $f_S / (2\, N_C)$, where $N_{FFT,S}$ is the number of signal samples used by the FFT. At the highest positive frequency bin, the loss is maximum and is $\mathrm{sinc}(\pi /2)$, i.e. about 3.9 dB. To reduce this loss, N_C must be decreased. For instance, the loss will be reduced to about 0.9 dB if the maximum Doppler frequency corresponds to $f_S / (4\, N_C)$. The price paid in this case is the extra computation of the frequency bins outside the search space, which will not be subsequently used.

A second drawback is the extra loss (called scalloping loss) that occurs when the Doppler frequency falls in between two FFT bins. This loss can attain a maximum value of about 3.9 dB if the signal frequency is exactly in the middle of two FFT bins (for more details the reader is referred to [9, 10]). A simple but efficient mitigation solution is the use of zero padding. For example, padding the signal with as many zeros as there are samples (i.e. doubling the number of bins, $N_{FFT} = 2\, N_{FFT,S}$) will bring the maximum loss down to about 0.9 dB. This is the method that we consider in this paper (note that other techniques exist, such as windowing [11]).

The third drawback is the loss linked to the mismatch between the replica code chipping rate and the received code

chipping rate which is also affected by the Doppler effect. Indeed, several carrier frequencies are tested through the FFT whereas there is only one code chipping rate tested. For example, with the GPS L1 C/A signal, if the carrier frequency is shifted by 5 kHz due to the Doppler effect, the code chipping rate will be shifted by about 3.25 chip/s (i.e., 5000×1.023/1575.42). This means that the code replica and the received code will shift by about 3.25 chips every second, or one quarter of a chip every 77 ms. To reduce this effect, the frequency search space must be cut into several smaller spaces [12].

3) Parallel Code-phase Search

A second solution to reduce the acquisition time is to parallelize the search in the code-phase space. Thanks to the relationship between the convolution in the time domain and the multiplication in the Fourier domain, it is possible to compute the circular cross-correlation of two signals, $s[n]$ and $c[n]$, using the FFT and the Inverse FFT (IFFT) as shown by (3).

$$r_{sc}[n] = \mathrm{IFFT}\left\{\mathrm{FFT}\left[s[n]\right]\mathrm{FFT}\left[c[n]\right]^{*}\right\} \qquad (3)$$

The corresponding architecture is illustrated in Fig. 4 where the FFTs and the IFFT are typically performed on one period of the code (corresponding to N_{CB} samples here, which is defined in Section II.C) [13]. Note that multiple periods of the code could be used, but this would not bring any advantage. Moreover, using lengths shorter than one code period with a technique such as overlap-and-add is also possible, but adds complexity [14] (which is why it is not considered in this paper). The IFFT results are then added to perform the coherent integration. The magnitude computation and the non-coherent integration are then performed. Except for the carrier replica multiplication, all the operations are performed on vectors. In this case, only the different carrier frequencies are tested sequentially. Looking at Fig. 1, this means that the rows are tested one after the other.

The advantage of this architecture is its very high gain in processing time, since it is equivalent to have as many correlators as there are points in the FFT (each point being equivalent to a code bin).

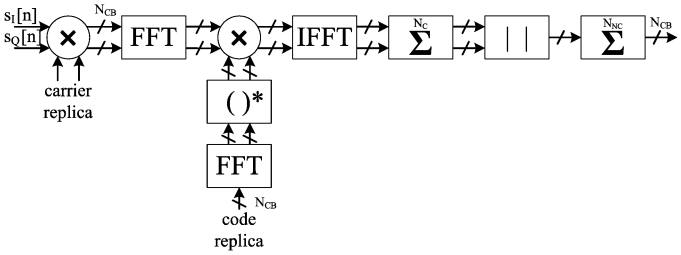

Fig. 4. Diagram of Parallel Code-phase Search (PCS) acquisition

A first drawback of this architecture is the limited number of choices for the sampling frequency. Indeed, the usual radix-2 FFT algorithm can be performed only with sequences of 2^N points. In this case, zero padding can be used with the following constraint : the number of points should be at least twice the number of samples in one period of the code, to ensure that there will not be any loss [15]. For the GPS L1 C/A signal with a sampling frequency of 4 MHz, the signal should be padded to obtain sequences of 8192 points and the replica should contain two periods (padding only to 4096 points would potentially result in losses by spreading the peak over several code bins if the zeros are not padded to the beginning or the end of the code period 16]). This of course increases the complexity of the FFT. This problem can also be solved by other means, including performing a traditional sample rate conversion [17], using GNSS specific signal compression algorithms [18, 19], or using other FFT algorithms that can be applied to sequences whose length is not a power of two [20]. However, these algorithms are typically more difficult to implement.

A second drawback of this architecture is the relatively large size of the FFT, which depends on the sampling frequency and on the code length, and can be too big for some implementations. This problem can be circumvented by using a smaller FFT with the overlap-and-add technique [14], performing decimation [17], or using a signal compression algorithm [18, 19]. In this paper, we consider sampling frequencies for which the number of samples per code period is a power of two, and no signal decimation or compression. However any other choice can be easily handled.

The third drawback is a loss linked to a potential bit transition due to data or secondary code inside the input sequence used for the correlation, since the sequence does not start necessarily at the first chip of the code. This problem does not occur for the two other architectures, which always start the integration at the first chip of the code. It can be resolved by doubling the FFT size and zero-padding the code replica, or with other algorithms [21].

4) Integration Time

In the three architectures, there is an accumulator involved in the coherent integration and an accumulator for the non-coherent integration. The number of samples used by the non-coherent accumulator for one integration and dump for one bin, N_{NC}, is identical for the three architectures. However, the number of samples used by the coherent accumulator for one integration and dump for one bin, N_C, is different. Therefore, to make a distinction between the architectures, we define $N_C \in \{N_{C,SS}, N_{C,PFS}, N_{C,PCS}\}$.

In the SS architecture, the coherent integration is performed only by the coherent accumulator, consequently $N_{C,SS}$ corresponds to the number of samples during the coherent integration time and is defined as

$$N_{C,SS} = T_C f_S = \frac{T_C}{T_S} \qquad (4)$$

where T_C is the coherent integration time in seconds, f_S the sampling frequency in hertz, and T_S the sampling period in seconds.

In the PFS architecture, the coherent integration is performed in two steps, with the coherent accumulator followed by the FFT. $N_{C,PFS}$ can thus be expressed as

$$N_{C,PFS} = \frac{T_C f_S}{N_{FFT,S}} = \frac{T_C}{N_{FFT,S} T_S} \quad (5)$$

where $N_{FFT,S}$ is the number of signal samples used in the FFT. The values of $N_{C,PFS}$ and $N_{FFT,S}$ are chosen according to the size of the frequency search space and the integration time to minimize the losses previously described [9, 10]. Note that their product is equal to $N_{C,SS}$.

In the PCS architecture, the coherent integration is also performed in two steps, with the FFTs performing the correlation over one period of the PRN code (T_{code}), followed by the coherent accumulator. $N_{C,PCS}$ can thus be written as

$$N_{C,PCS} = \frac{T_C}{T_{code}}. \quad (6)$$

To clarify this, we consider an example. For the stand-alone case with the GPS L1 C/A signal (T_{code} = 1 ms), a coherent integration time T_C = 10 ms and a sampling frequency f_S = 4.096 MHz, we obtain the values listed in Table I. It can be seen that the values are very different : relatively high for the SS architecture, small for the PCS architecture, and in between for the PFS architecture.

TABLE I
COHERENT INTEGRATION PARAMETERS EXAMPLE

Architecture	Coherent Parameters
SS	N_C = 40 960
PFS	N_C = 160 ; $N_{FFT,S}$ = 256
PCS	N_C = 10

During the coherent integration time, there are thus 40,960 samples to process. The coherent accumulator of the SS will perform thus 40959 accumulations, the one of the PFS will perform 256×159=40704 accumulations, and the one of the PCS will perform 9×4096= 36864 accumulations.

The values of N_C and N_{NC} also have an impact on the design, since they influence the resolution of the output of the accumulators (the details are provided in appendices).

5) Summary

A simple summary of the complexity and search parallelism of the three architectures depicted in Figs. 2, 3 and 4 is given in Table II.

TABLE II
ACQUISITION ARCHITECTURES COMPARISON

Architecture	Complexity	Search Parallelism
SS	low	no
PFS	medium	medium
PCS	high	high

We note that PCS provides the highest parallelism. Therefore, it is not surprising that most research groups that have developed hardware GNSS receivers in the past years have selected it for implementation. However, despite providing the best parallelism, it is also the most demanding in terms of complexity or hardware resources. Consequently, it may be important to consider the other architectures as well, as they may provide a better trade-off between implementation complexity and search parallelism. Towards this goal, we explore here hardware duplication and different optimizations for each architecture to enhance its search parallelism, and compare the architectures assuming a given hardware resource usage.

C. Acquisition Parameters

Now, we present the different parameters that have an impact on acquisition. They are classified into two classes : 1) the primary parameters that are given by the context (application and hardware); and 2) the secondary parameters that are derived from the primary ones. These parameters finally lead to the computation of the acquisition time. The relationship between these parameters is depicted in Fig. 5, where a solid line means that the link is present for all the architectures; a dashed line means that the link is present only for the PCS architecture; and a dotted line means that the link is present only for the PFS architecture.

1) Primary Parameters

a) Code-phase Resolution Δ_C

This is the minimum step between two tested code-phases. It depends on the signal modulation and the precision required. With the PCS architecture, the resolution is typically linked to the sampling frequency and corresponds to one sample, but it can be different if a preprocessing step, like decimation or averaging, has been used. It impacts the number of code bins and the integration time [2]. It is denoted as Δ_C and can be expressed in chip or in sample, i.e. $\Delta_C \in \{\Delta_{C,chip}, \Delta_{C,sample}\}$.

b) Sensitivity

This is the desired minimum received signal power in dBm that can be detected. The carrier-to-noise ratio, denoted C/N_0 and expressed in dB Hz, can be used instead equivalently. The sensitivity impacts the integration time [2].

c) Frequency Search Space f_{SS}

This is the frequency range where signals can be found. It impacts the number of frequency bins. It is denoted as f_{SS} and expressed in Hz.

d) FPGA

This is the target chip. The family of the FPGA impacts the maximum FPGA frequency, and the resources inside the FPGA impacts the parallelization that can be applied. More details about FPGAs are provided in Section III.A.

e) FPGA Frequency f_{FPGA}

This is the frequency of the clock inside the FPGA at which the acquisition channel runs. It directly impacts the FPGA processing gain. The higher it is, the faster the processing will be, as detailed in Section III.B. It is denoted as f_{FPGA} and expressed in Hz.

f) Sampling Frequency f_S

This is the frequency at which the signal is sampled. It impacts the parallelization that can be applied and the FPGA processing gain since, the higher it is, the more data there will be to process. With the PCS architecture, it can also impact the

code-phase resolution. It is denoted as f_S and expressed in Hz.

2) Secondary Parameters

a) Number of Code Bins N_{CB}

This is the number of bins to test in the code-phase space. It is denoted as N_{CB}, is without units and is defined as

$$N_{CB} = \frac{N_{chip}}{\Delta_{C,chip}} = \frac{N_{sample}}{\Delta_{C,sample}} \quad (7)$$

where N_{chip} is the number of chips and N_{sample} the number of samples in the code-phase search space. When there is no assistance to provide fine time information, the code-phase search space corresponds to the entire PRN code.

b) Integration Time T_C and T_T

There are two components for this parameter, the coherent integration time and the total integration time (which includes the non-coherent additions). They are denoted as T_C and T_T, respectively, and expressed in seconds. They can be computed using the detailed method presented in [2]. They have an impact on several parameters. The coherent integration time impacts the frequency resolution as described in the next paragraph. The total integration time directly impacts the acquisition time, since the quantity of data to process is proportional to it. Finally, they both influence the parallelization that can be applied because they impact the resolution of the signals in several functions, such as in the coherent and non-coherent accumulators.

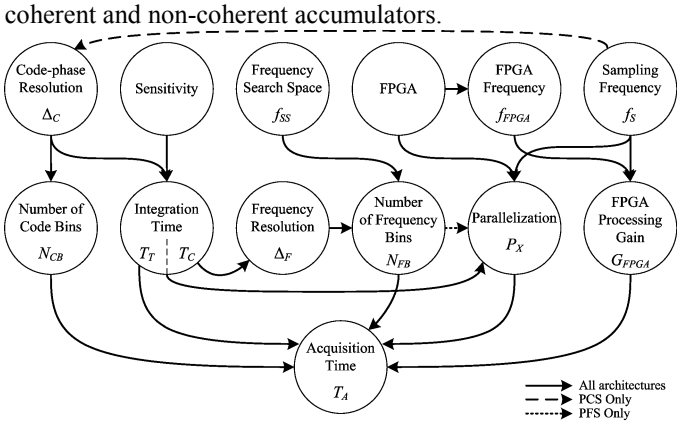

Fig. 5. Diagram of acquisition parameters

c) Frequency Resolution Δ_F

This is the minimum step between two frequencies tested. It is linked to the coherent integration time T_C, denoted as Δ_F and expressed in Hz.

As described in Section II, after coherent integration over a time T_C, the result is shaped as a sinc function whose width is inversely proportional to T_C and reaches zero at $1 / T_C$. Most of the time, Δ_F is chosen as $2 / (3\ T_C)$ or $1 / (2\ T_C)$. The second case leads to more bins to test but as the maximum loss is lower, the total integration time needed is also lower, and at the end both values give approximately the same averaged performance.

In the PFS architecture, applying the FFT on the signal without zero-padding is equivalent to having a frequency resolution of $1 / T_C$, while zero-padding the signal with as many zeros as there are samples is equivalent to having a frequency resolution of $1 / (2\ T_C)$.

For consistency between the architectures, we will thus consider a resolution of $\Delta_F = 1 / (2\ T_C)$ in the following.

d) Number of Frequency Bins N_{FB}

This is the number of bins to test in the frequency search space f_{SS}. It is denoted as N_{FB}, without units and defined as

$$N_{FB} = 2\left\lceil \frac{f_{SS} - \Delta_F}{2\,\Delta_F} \right\rceil + 1. \quad (8)$$

This formula allows a frequency bin centered on a desired frequency and the same number of frequency bins to test above and below this frequency. With the value previously chosen for Δ_F, (8) can be written as

$$N_{FB} = 2\left\lceil f_{SS}\ T_C - \frac{1}{2} \right\rceil + 1. \quad (9)$$

This parameter can impact the parallelization applied in the PFS architecture if the frequency space is searched entirely, since it corresponds to the minimum number of points of the FFT.

The number of cells of the search space is given by the number of the code bins times the number of frequency bins, i.e. $N_{cell} = N_{CB}\ N_{FB}$.

e) Parallelization P_X

The comparison between the architectures will be based on this parameter denoted as P_X where X denotes the architecture (SS, PFS or PCS). It corresponds to the number of cells of the search space that are tested simultaneously and includes the parallelization brought by the FFT and by the duplication of the elements. It depends on many other parameters and will be determined in Section III.G.

f) FPGA Processing Gain G_{FPGA}

This is the gain in processing time given by the ratio of the FPGA frequency to the sampling frequency. It is denoted as G_{FPGA}, without units and defined as

$$G_{FPGA} = \frac{f_{FPGA}}{f_S}. \quad (10)$$

3) Acquisition Time

a) Search Time of the Full Time-Frequency Space $T_{F,X}$

The time to explore the whole time-frequency search space is given by

$$T_{F,X} = \frac{\beta\, T_T}{G_{FPGA}} \frac{N_{CB}\, N_{FB}}{P_X} \quad (11)$$

where $\beta = \begin{cases} 2, & \text{for alternate half-bits method} \\ 1, & \text{for other methods} \end{cases}$.

The alternate half-bits method consists of creating two sets of alternate portions of the signal in order to have one without a data bit transition (a transition can occur when one bit of data lasts more than a period of the PRN code) [22]. This requires doubling the length of the signal used to keep the same SNR after the integration. There are two other methods that do not require an extra length of signal. The first, called the full-bits method, consists of integrating the signal for all the

possibilities for the data bit transition (e.g. 20 for GPS L1 C/A) so that at least one will be free of data transitions [22]. The second method simply ignores the data bit transition and counts it as a loss in the SNR balance sheet [2].

$\beta\ T_T\ /\ G_{FPGA}$ corresponds to the time to process the signal recorded at the FPGA rate. In order to explore the entire search space, the signal is processed several times, depending on the number of cells making up the search space and the implemented parallelization.

Note that the time to load a new code, or to modify the carrier and code frequencies on the channel, and the latency in the processing are not taken into account in this formula. Indeed, the loading time is very small, typically on the order of dozens of cycles. The latency is mainly due to the FFTs and corresponds to the size of the elements, i.e. a few thousands of clock cycles, whereas the input signal used is typically composed of hundreds of thousands of samples if high-sensitivity is intended. Therefore the latency represents only a very low percentage of $T_{F,X}$.

b) Mean Acquisition Time $\overline{T_{A,X}}$

Of course, it is usually not necessary to explore the whole search space to find a satellite. Regarding the code-phase space, the phase of the code is completely random and follows a uniform distribution. Regarding the frequency space, the distribution of the Doppler frequency depends on the context, i.e. the user position (latitude/longitude) as well as the constellation. This implies that the global distribution of Doppler frequencies does not allow a particular strategy. Consequently its distribution is often considered as uniform and the search starts with the frequencies near the one expected without a Doppler effect (corresponding to the highest elevation satellites) and finishes with the farthest frequencies (corresponding to the lowest elevation satellites). Note that if assistance is available (for code or frequency), the uniform distribution may no longer be applicable.

The last parameters involved in the acquisition time are the probabilities of detection and false alarm. Indeed, it is possible to miss a satellite after having explored the whole search space (no detection), or to have seemingly found a satellite where one is not (false alarm), which leads to bad tracking and re-acquisition after a certain time.

The detection test is performed when the output of the architecture is available, i.e. when a portion of the search space (portion of a row for the SS, several rows for the PCS, and a rectangular area for the PFS) is available. The mean acquisition time $\overline{T_{A,X}}$ for N_{sat} satellites, taking into account signal buffering, is adapted from [23] and can be approximated by

$$\overline{T_{A,X}} = \beta T_T + \frac{2-P_D}{2P_D}\left[1+k\,G_{FPGA}\left(1-\left(1-P_{FA}\right)^{P_X}\right)\right]T_{F,X}\,N_{sat} \quad (12)$$

where X denotes the architecture (SS, PFS or PCS), P_D the probability of detection, P_{FA} the probability of false alarm, and k a penalty factor characterizing the time to detect a false alarm. The reader can also refer to [24, 25] for more details on acquisition time and probabilities.

III. Detailed Analysis of Architectures

In this section, the hardware implementation of each architecture is analyzed in detail. First, some basic information regarding FPGAs is provided. Second, buffering of the input signal and high frequency processing are presented. Third, possible ways to parallelize processes are explored for each architecture. After this, some important points regarding the FFT are discussed, and finally the determination of the parallelization and of the search time of the full space is provided.

A. FPGA Considerations

An FPGA is a programmable device containing three main types of elements :

- Logical block : This is a small block containing a Look-Up Table (LUT) allowing the creation of logic functions, a full adder, and one or several registers. This basis block is different for each manufacturer and even between some FPGA families.
- Memory block : This is a memory of small size (typically between 0.5 and 128 Kibit), consisting of multiple ports.
- Digital Signal Processing (DSP) block : This is a block containing hardware multipliers (typically 18 × 18 bits).

To optimize the implementation, the usage of these elements has to be balanced. It is relatively easy to estimate resource usage for the memory and DSP blocks because it is easy to determine the number of bits and the number of multiplications required in a system. However, for the logical blocks, resource usage is more difficult to estimate for several reasons. 1) These blocks contain logic and registers, and a block can use one or the other or both, depending on the function implemented. For example, a counter or an accumulator needs as many registers as it has bits and this gives the number of logical blocks necessary, whereas for functions like multiplexing or magnitude computation, the number of blocks required is not so straightforward and empirical formulas have to be used. 2) The compilation tools perform various optimizations that can affect the final implementation. 3) These blocks are different according to the manufacturer or even between different families with different performances, which means that it is not possible to make a universal estimation.

In this paper, we base our estimate on FPGAs from Altera, first on the Cyclone series which have Logical Element (LE) basis blocks, and then on the Stratix series which have Adaptive Logic Module (ALM) basis blocks. The same estimation can be performed with FPGAs from other manufacturers, and approximate conversions can be applied between them although this is not undertaken here. The details of the resource estimates of the architectures are given in Appendix 1 in order to not overload the body of this article.

B. Signal Buffering

The simplest way to perform the correlation is to process the streaming signal at the sampling rate. However, another well-known way to proceed is to first buffer the streaming signal before processing it fast enough to still allow real-time processing, as depicted in Fig. 6. This method allows a great gain in processing time. For example if the sampling frequency is 5 MHz and the FPGA frequency is 200 MHz, the processing time will be divided by 40.

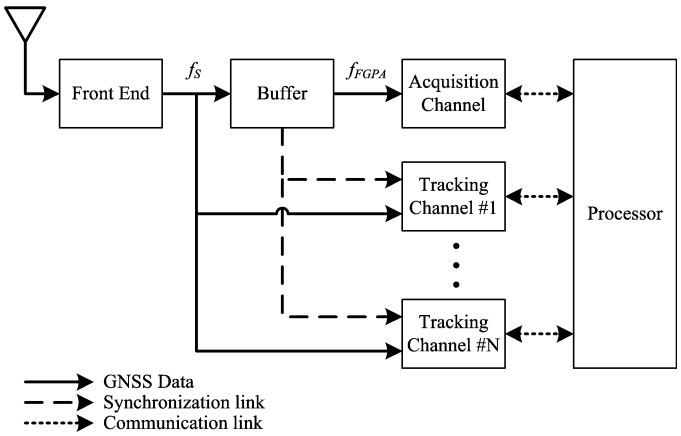

Fig. 6. Overview of a GNSS receiver using signal buffering

Signal buffering does not necessarily affect the tracking channels that process the signal at the sampling rate. However, in some cases, a tracking channel can also work at higher frequency in order to process the signals from different satellites and to save hardware resources.

C. Serial Search

The SS architecture depicted in Fig. 2 tests a single code-phase/carrier frequency bin at once. In order to improve processing speed the blocks can be duplicated in order to have several branches (denoted as N_B in the following) testing different frequencies or different code-phases at the same time.

The code and carrier replicas are generated using a Numerically Controlled Oscillator (NCO), which is a counter in which the increment specifies the output frequency. The size of this counter is typically 32 bits, which requires 64 registers (32 for the increment value and 32 for the counter value). Generating different carrier frequencies requires as many NCOs as there are frequencies. Generating shifted versions of the code replica demands only one NCO and one register per delay. Consequently it is clearly more efficient to test several code-phases rather than several frequencies. The implementation of such duplication, which is similar to that in [26] with the addition of the multiplexer and non-coherent integration, is depicted in Fig. 7. At the bottom of the figure, the data rate is shown; a bar above a value indicates an average.

The mixers as well as the coherent accumulators run at the frequency f_{FPGA}. The rate at the output of the coherent accumulators is then divided by $N_C = T_C f_S$. Since the accumulation of the different accumulators starts and ends at different clock cycles (an accumulation always starts at the first sample of the code), multiplexing the next blocks, i.e. magnitude computation and non-coherent accumulation, is possible. Note that the non-coherent accumulator's input and output are not shown as vectors since the samples arrive serially, i.e. at each clock cycle. To differentiate this block from the traditional accumulator's block, we added the letter M in the bottom right corner of the block, (M for memory, since it uses memory as detailed in the next paragraph). The same applies for Figs. 9, 10 and 11.

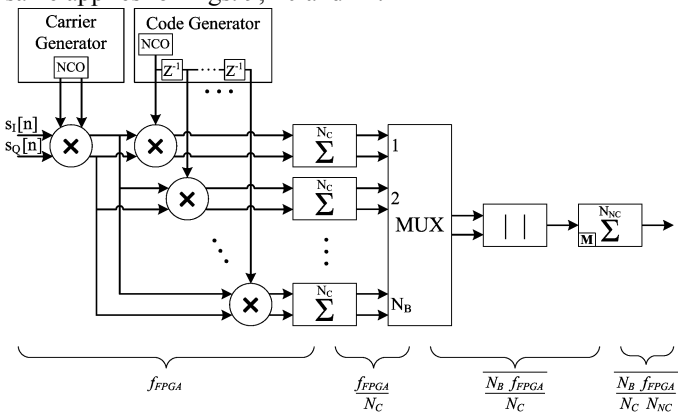

Fig. 7. Implementation of the Serial Search (SS) architecture with duplication

To study more deeply the implementation, we note that although the mixers perform a multiplication, they are implemented with logical blocks instead of DSP blocks. Indeed, the signals are quantized with few bits (typically two or three), and mixing with the code replica consists simply of changing the sign of the signal (except if the code has more than two levels). The coherent accumulators are classical adders implemented with logical blocks. It is possible to optimize the implementation by fusing a code mixer and an accumulator into an accumulator that can add or subtract the input value according to the value of the code. This optimization is discussed in Appendix 1. The multiplexer is implemented with logical blocks. The magnitude computation can be performed with different algorithms, the simplest being the Robertson approximation [17]. Finally, the non-coherent accumulator is implemented with memory blocks in order to save logical blocks, using only one adder and one multiplexer, as depicted in Fig. 8. Each address is associated with a sample of the cross-correlation function, and is written and read $N_{NC} - 1$ times to perform the accumulation. The memory has thus N_B addresses. With a memory-based accumulator, the data rate is reduced in average only, because there are N_{NC} times fewer samples in the output than in the input, but the output rate is the same as the input rate.

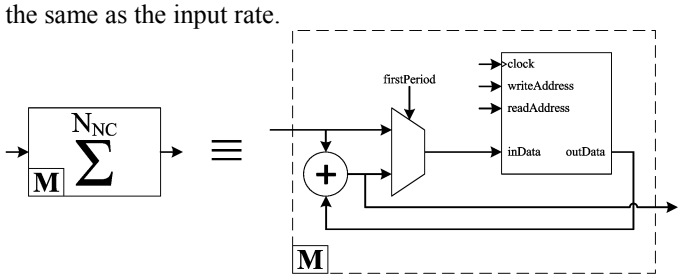

Fig. 8. Diagram of a memory-based accumulator

In the SS architecture, the duplicated elements are the code mixer and the coherent accumulator, and the multiplexer is proportional to the number of branches. The most-used resources are clearly the logical blocks, as the memory is used only with the non-coherent accumulator and to store the PRN code, and the DSP blocks are not used at all.

D. Parallel Frequency Search

Following the same idea, the structure depicted in Fig. 3 can be duplicated in order to test several code-phases at the same time. Such implementation of the PFS architecture is depicted in Fig. 9. The carrier and code NCOs and mixers, the coherent accumulators and the multiplexer are identical to the ones seen previously. Remember that in this architecture the coherent accumulator does not accumulate for the entire coherent integration time but as defined by (5). Then there is a particular buffer, which has a writing order different from the reading order. This is because after the multiplexer there are first the first points of each branch, then the second points of each branch, etc., whereas the FFT should be first fed with all the points of the first branch, then with all the points of the second branch, etc. Moreover, since data can be written at addresses not yet read, it is necessary to use two buffers, one being read while the other is written to, which alternate their roles (a ping-pong buffer). Next is the FFT block which uses logical, DSP and memory blocks. As mentioned in Section II, according to the N_C and $N_{FFT,S}$ values selected, only a portion of the FFT bins may be necessary to cover the search space. The number of bins kept is denoted N_{FT}, which is equal to N_{FB} only if the entire frequency search space is covered by the FFT. The rate after the FFT is thus reduced by N_{FFT} / N_{FT}. Finally there are the magnitude computation and the non-coherent accumulator based on memory blocks. The memory inside the non-coherent accumulator has N_B N_{FT} addresses in this case.

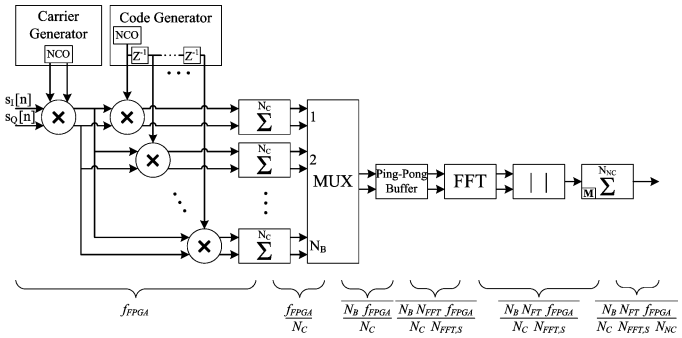

Fig. 9. Implementation of the Parallel Frequency Search (PFS) architecture with duplication

In this implementation the resource usage of the logical elements is relatively similar to the SS architecture because the accumulators are a little bit smaller and there is just one supplementary FFT, but the memory is used far more. However, there are two limitations with the direct implementation depicted in Fig. 9.

First, the number of branches is limited by the number of accumulations performed by the coherent accumulator. Indeed after the accumulator stage the rate is divided by N_C, and after the multiplexer the rate is multiplied by N_B. Since the rate after the multiplexer cannot be superior to the initial FPGA rate, $N_B \leq N_C$. However, this limit may be easily circumvented by implementing several multiplexer chains (multiplexer, FFT, etc.).

The second limitation is that if zero-padding is used, which is common, the data rate after the FFT is superior to the data rate before. Consequently, in one multiplexer chain, the number of branches is limited according to the following equation,

$$N_B \leq N_C \frac{N_{FFT,S}}{N_{FFT,S} + N_{FFT,Z}} \leq N_C \frac{N_{FFT,S}}{N_{FFT}} \quad (13)$$

where $N_{FFT,S}$ is the number of points of signal used for the FFT and $N_{FFT,Z}$ the number of zeros padded (the total number of points of the FFT being $N_{FFT} = N_{FFT,S} + N_{FFT,Z}$). This limit can be circumvented in the same manner as before by implementing several multiplexer chains.

E. Parallel Code-phase Search

Still following the same idea, the structure depicted in Fig. 4 can be duplicated to test several carrier frequencies at the same time since all the code-phases are already tested. Such implementation of the PCS architecture is depicted in Fig. 10. The carrier and code NCOs and the carrier mixer are identical to those seen previously. The FFT block is similar to the one in the PFS architecture, but larger. The complex multipliers in the frequency domain use DSP blocks. And now, both accumulators (coherent and non-coherent) are implemented with memory blocks, which have N_{CB} addresses in this case. In this architecture, no multiplexing can be performed since all the data operate at the FPGA frequency, even after the coherent accumulator where the data rate is reduced in average only. But the accumulators' addressing can be shared since the FFTs start and finish at the same time.

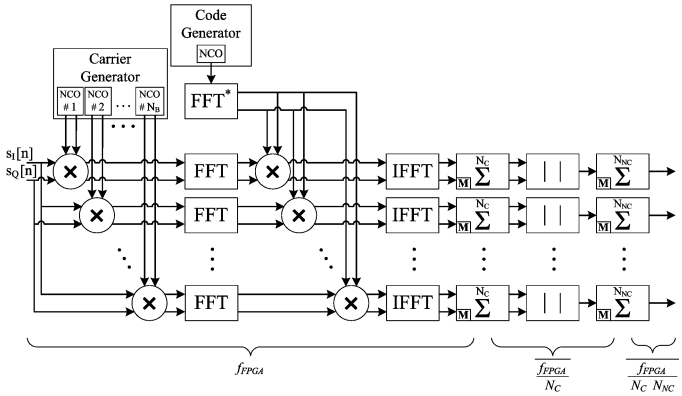

Fig. 10. Implementation of the Parallel Code-phase Search (PCS) architecture with duplication

This architecture is the one that best balances the use of the different elements, since it uses the logical blocks and DSP

blocks with the FFT, and the memory blocks with the FFT and the accumulators.

The problem described with the PFS regarding the influence of the Doppler effect on the code is present here also, but to a lesser extent, since several carrier frequencies are tested and only one code chipping rate is generated. Indeed, the carrier frequencies generated at the same time in this case generally cover just a portion of the frequency search space and are relatively close to each other. For example, still considering the GPS L1 C/A signal, if the carrier frequencies cover a Doppler range of ± 150 Hz, the maximum Doppler shift on the code chipping rate will be about 0.097 chip/s, which implies a shift of about one quarter of a chip every 2.6 seconds only. Moreover, here the effect can be removed by generating a code for each carrier frequency tested, at the expense of additional FFTs, or more usually by applying a correction during the coherent accumulation stage (shift of the IFFT outputs or multiplication by a carrier in the frequency domain) [27].

An optimization of this architecture is possible if the frequency search space is wide enough. Instead of multiplying the input signal by different carrier replicas and performing several FFTs, only one carrier replica and two FFTs (one for the input signal, one for the code replica) can be used, and the multiplication by the different carriers is replaced by shifts of the FFT output. A shift of one sample is equivalent to a multiplication by a 1 kHz carrier if the FFT length is 1 ms thanks to the DFT shifting theorem [17]. For example, considering five branches, it means that it would be possible to test simultaneously the carrier frequency bins {0, 1000, -1000, 2000, -2000} Hz, and if no signal is found, to continue with the followings, i.e., {50, 1050, -950, 2050, -1950} Hz, etc. With this optimization the architecture would require 2 FFTs instead of N_B + 1, and still N_B IFFT, as shown in Fig. 11. This optimization is also considered in the example application in Section IV, referred as PCS* architecture.

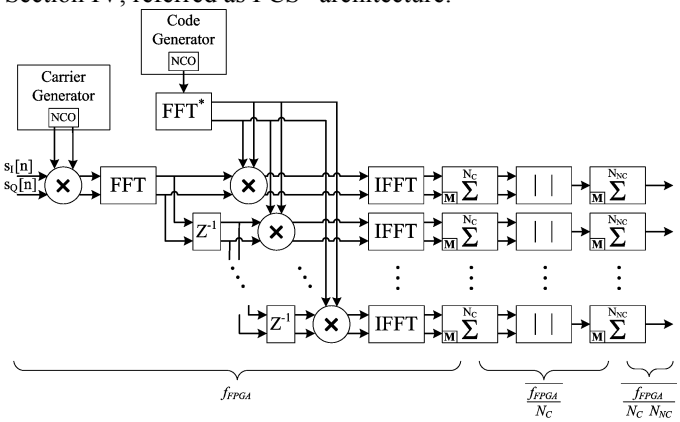

Fig. 11. Implementation of the Parallel Code-phase Search (PCS) architecture with duplication and shifting in frequency domain (referred to as "PCS*" in the text and Table V to differentiate it from Fig. 10)

F. FFT Considerations

The FFT algorithm can handle data in normal order or bit-reversed order, as shown in Table III [17]. Traditionally the order of the input and output are the same, but the order should be different to minimize resource usage and latency.

For the PFS architecture, it is not important if the data at the output of the FFT are in the bit-reversed order because reordering can be done through addressing of the non-coherent accumulator memory. Since it consists only of reversing the bits it costs nothing in terms of resources. For the PCS architecture, the reordering can be naturally done because the FFT is followed by an IFFT. This is also valid for software receivers.

TABLE III
DATA ORDER OF AN 8-POINT SIGNAL

Normal order of index n	Bit-reversed order of index n
0 (000)	0 (000)
1 (001)	4 (100)
2 (010)	2 (010)
3 (011)	6 (110)
4 (100)	1 (001)
5 (101)	5 (101)
6 (110)	3 (011)
7 (111)	7 (111)

The implementation of an FFT is very flexible, with a large number of parameters including data and twiddle factor resolution, arithmetic type (integer or block floating point), fixed or variable length, etc. These have different impacts on the logic, memory and DSP usage, and should be well studied for optimal performance for each specific context.

G. Parallelization

Now that the architectures have been analyzed, we can compute the parallelization and the search time of the full space for each one, which then provides the mean acquisition time through (12).

1) Serial Search

Parallelization of the SS architecture comes from hardware duplication and corresponds to the number of branches implemented.

$$P_{SS} = N_{B,SS} \tag{14}$$

Applying this to (11) gives

$$T_{F,SS} = \frac{\beta T_T}{G_{FPGA}} \frac{N_{CB} N_{FB}}{N_{B,SS}}. \tag{15}$$

2) Parallel Frequency Search

Regarding the PFS architecture, there are two components to parallelization. The first comes from hardware duplication like for the SS architecture and corresponds to the number of branches implemented. The second comes from the FFT that allows the search of several or all the frequency bins at one time.

$$P_{PFS} = N_{B,PFS} N_{FT} \tag{16}$$

Applying this to (11) gives

$$T_{F,PFS} = \frac{\beta T_T}{G_{FPGA}} \frac{N_{CB} N_{FB}}{N_{B,PFS} N_{FT}}. \quad (17)$$

3) Parallel Code-phase Search

In the same way, the PCS architecture also has two components to its parallelization. The first comes from hardware duplication like for the other architectures and corresponds to the number of branches implemented. The second comes from the FFT that allows the search of all the code bins at one time.

$$P_{PCS} = N_{B,PCS} N_{CB} \quad (18)$$

Applying this to (11) gives

$$T_{F,PCS} = \frac{\beta T_T}{G_{FPGA}} \frac{N_{FB}}{N_{B,PCS}}. \quad (19)$$

IV. Results

A. Application

Now that the different parameters and implementations have been described, the performance of the three architectures is compared through an example. A low-cost FPGA (Altera Cyclone III EP3C120) and a high-end FPGA (Altera Stratix III EP3SE260) are investigated. For each architecture, we select the implementation that maximizes the use of the FPGA resources. Note that it is not possible to entirely fill an FPGA due to routing constraints; we thus consider the use of 85 % of the logical blocks inside the FPGAs [28].

The analysis has been applied to two cases using the GPS L1 C/A signal for two cases : a stand-alone case where the receiver has no a priori information, and an assisted case where the receiver has a priori information on the Doppler frequency of the satellites, which reduces the frequency search space [16]. A sensitivity of – 150 dBm is assumed, since this is the start of high sensitivity; the required integration times are obtained with the method from [2], applying the half-bit method for managing the data bit transitions (β = 2). A sampling frequency of 4.096 MHz is a good compromise between induced complexity and accuracy. The FPGA frequencies selected are multiples of the sampling frequency, and realistic values obtained from real designs. All the acquisition parameters are given in Table IV.

The search space is composed of 905 216 cells in the stand-alone case, and of 118 784 cells in the assisted case.

TABLE IV
Parameters Selected for the Acquisition of GPS L1 C/A Signals

Primary Parameters	Secondary Parameters
Δ_C = 1 sample	N_{CB} = 4096
Sensitivity = – 150 dBm	T_C = 10 ms T_T = 400 ms
f_{SS} = 11 020 Hz (stand-alone) = 1360 Hz (assisted)	Δ_F = 50 Hz
FPGA = Altera EP3C120 = Altera EP3SE260	N_{FB} = 221 (stand-alone) = 29 (assisted)
f_{FPGA} = 98.304 MHz (EP3C120) = 196.608 MHz (EP3SE260)	P_X = (cf Table V)
f_S = 4.096 MHz	G_{FPGA} = 24 (EP3C120) = 48 (EP3SE260)

With such a long integration time, the maximum error in the code chipping rate allowed, to have a shift smaller than half a sample, is about 0.156 chip/s. The PFS can thus search only ± 240 Hz of the frequency search space simultaneously, i.e. N_{FT} = 11.

The details of the calculations are provided in Appendix 2, and results for the number of branches, parallelization and search time of the full space are given in Table V. The number of branches gives the degree of duplication in the architectures depicted in Figs. 7, 9, 10 and 11. The parallelization is the number of cells tested simultaneously and is used to compare the architectures. Besides this value, the percentage of cells tested over the total number of cells of the time-frequency search space is given in parenthesis. The search time of the full space is maybe more meaningful for GNSS users since it gives an idea of the processing time, and it can also be used to compare the architectures.

TABLE V
Implementation Results for the Number of Branches, Parallelization and Search Time of the Full Time-Frequency Space, for the GPS L1 C/A Signal.
The values in parentheses represent the percentage of the time-frequency space searched Simultaneously (100 P_X / N_{CELL}).

Parameter	Low-cost FPGA Altera EP3C120		High-end FPGA Altera EP3SE260	
	Assisted Case	Stand-alone Case	Assisted Case	Stand-alone Case
$N_{B,SS}$	971		2911	
$N_{B,PFS}$	1095		3385	
$N_{B,PCS}$	2		8	
$N_{B,PCS}^*$	-	4	-	11
P_{SS}	971 (0.8 %)	971 (0.1 %)	2911 (2.5 %)	2911 (0.3 %)
P_{PFS}	12 045 (10.1 %)	12 045 (1.3 %)	37 235 (31.3 %)	37 235 (4.1 %)
P_{PCS}	8192 (6.9 %)	8192 (0.9 %)	32 768 (27.6 %)	32 768 (3.6 %)
P_{PCS}^*	-	16 384 (1.8 %)	-	45 056 (5.0 %)
$T_{F,SS}$ (ms)	4078	31 075	680.1	5183
$T_{F,PFS}$ (ms)	328.7	2505	53.17	405.2
$T_{F,PCS}$ (ms)	483.3	3683	60.42	460.4
$T_{F,PCS}^*$ (ms)	-	1842	-	334.8

The mean acquisition time for different numbers of satellites is depicted in Fig. 12 for the EP3SE260 FPGA, a probability of false alarm P_{FA} of 10^{-8} (common for high-sensitivity receivers [2]), which gives a probability of detection P_D of about 0.92, and a penalty factor k of 10.

From Table V and Fig. 12, it can be seen that the SS

architecture is the least efficient of the three; even with assistance the result is worse than for the other architectures in the stand-alone case. The PFS architecture is slightly more efficient than the PCS architecture. If the PCS architecture is optimized with shift in the frequency domain (PCS* architecture), then it becomes slightly better than PFS for the stand-alone case. In the assisted case, most of the frequencies that can be tested through the different branches fall outside of the frequency search space, and are thus useless.

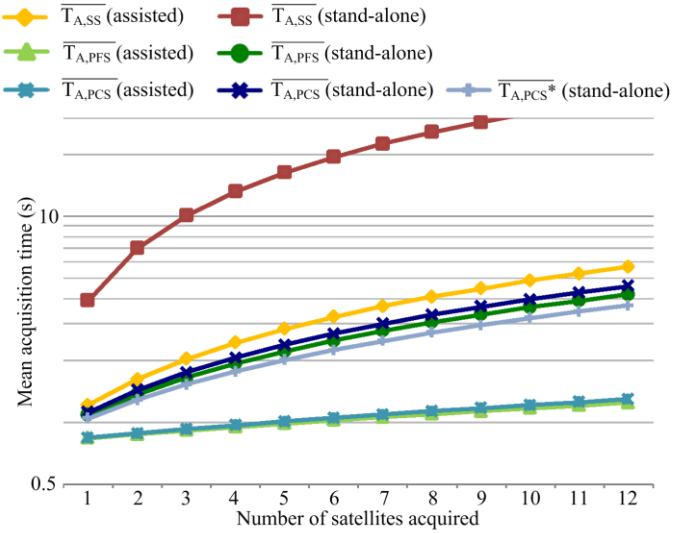

Fig. 12. Mean acquisition time of GPS L1 C/A signals assuming the parameters of Table IV, a $P_{FA} = 10^{-8}$ and a penalty factor $k = 10$

B. *Observations*

1) *Why the PFS and PCS are better than the SS*

Looking at Figs. 7 and 9, we note that the SS and PFS architectures are identical on the left, except that the accumulators of the PFS are smaller since the integration length is smaller. However, the PFS has a supplementary FFT. The resource usage of the logical elements is then almost equivalent between the two architectures. In fact, the better efficiency of the PFS over the SS architecture arises because the PFS architecture takes advantage of the memory of the ping-pong buffer and the non-coherent accumulator, whereas the SS does so only for the non-coherent accumulator.

Regarding the superior performance of the PCS over the SS architecture, it is well known that performing a convolution using an FFT is more efficient than with traditional large filters [14].

2) *Comparison of the PFS with the PCS*

Looking at Figs. 9 and 10, the two architectures are now completely different. The first point about the PCS architecture is that a lot of DSP blocks are used with the FFTs due to their large size and their number. Although the FPGA used was very rich in DSP blocks, it can be the element limiting the duplication (cf. Appendix 2). The second point is that the resolution of the data inside the FFT in the PCS architecture needs to be higher than in the PFS because of the longer chain to compute the correlation (FFTs, multiplication and then IFFT), resulting in a propagation of the quantization errors. The third point is that in the PCS architecture the main part of the memory is used by the FFTs, the rest being reserved for the storage of data. By contrast, the memory used in the PFS architecture is almost only for storage, not for computation (except for the relatively small FFT), which means that more data can be stored. This explains the better efficiency of the PFS over the PCS architecture. However, if the optimized PCS can be used, several FFTs are saved, and in this case this architecture becomes more slightly efficient than the PFS.

Moreover, there are two points that make the PFS more flexible and attractive than the PCS. The first concerns the impact of the sampling frequency. With the PFS architecture, doubling the sampling frequency would result in adding one bit in the coherent accumulators, i.e. R+1 bits to store instead of R. Thus we can interpolate roughly by saying that keeping the same hardware resources, the number of branches would be divided by (R+1)/R. On the other hand, with the PCS architecture, doubling the sampling frequency would double the size of the FFT and of the accumulators; consequently, the number of branches would be divided by 2 (except if a resampling block is included in the acquisition channel). So the PFS is far less sensitive to sampling frequency than the PCS. The second point concerns the resolution of the code-phase space. It is imposed by the sampling frequency in the PCS architecture and can lead to a very high and not necessarily useful precision (unless a resampling has been performed), whereas the resolution can be freely chosen in the PFS architecture. Note that in our application we considered a sampling frequency with a number of samples per code period which is a power of two, and no signal decimation or compression. If signal decimation or compression was applied, the complexity of the architectures would be reduced, particularly for the PCS. On the contrary, without decimation or compression, selecting a sampling frequency that does not allow the direct use of an FFT algorithm will increase the complexity of the PCS architecture.

A little disadvantage of the PFS compared to the PCS is the sensitivity loss because of the integration preceding the FFT. This loss can reach 0.9 dB for the largest magnitude Doppler frequencies in our application.

The weakness of the PFS lies in its sensitivity to the Doppler effect on the code. If the code chipping rate was not altered, the entire frequency search space would be covered by the FFT, regardless of the total integration time used, and the PFS would be clearly better than the PCS. But in our application, where a relatively long total integration time is considered, the PFS searches only 11 frequency bins at the same time while the frequency space contains 221 bins in the stand-alone case. If we consider the new GNSS signals that use higher code frequencies, the effect will be amplified. If the shift is 3.25 Hz with a 1.023 MHz code, it will be of 32.5 Hz with a 10.23 MHz code. For the same integration time, the space covered by the FFT should thus be reduced in the same proportion to test only one or a few bins. The performance of

the PFS would then degrade and become closer to that of the SS, and thus be worse than the PCS.

3) Influence of FPGA

From Table V, we can see that despite the large differences in the absolute results for the two FPGAs, the ranking of the architectures is the same.

Generally, inside an FPGA family, the ratio between the different types of resources is very similar, i.e. using a bigger FPGA will provide an equivalent increase of the logical, memory and DSP blocks. Consequently for different FPGAs of the same family, we do not expect the ranking to change significantly.

Between different families, the ratios between logical and memory, as well as logical and DSP blocks, are different. For the same amount of logical blocks, a high-end FPGA will have more memory and DSP blocks than a low-cost FPGA. High-end FPGAs are consequently more suited for FFT-based architectures. However, this should not impact the ranking since the SS architecture is far inferior to the others in terms of performance. High-end FPGAs also accept a higher clock frequency, which improves the performance of all the architectures in the same manner.

V. Conclusion

In this paper, we have presented a framework to compare the main GNSS signals acquisition architectures on FPGAs. The implementations have been optimized towards achieving maximum parallelization for a single acquisition channel and fixed resources. It has been shown that the two FFT-based architectures are far more efficient than simple duplication of mixers and accumulators. Between these two architectures, considering the GPS L1 C/A signal with long integration times (10 ms of coherent, 400 ms of non-coherent, total of 800 ms of data due to half-bit method for managing the data bit transitions), the Parallel Frequency Search has been shown to provide slightly smaller mean acquisition time (3.62 s against 3.95 s for 10 satellites) and greater flexibility regarding the sampling frequency than the Parallel Code-phase Search. Nevertheless, if the frequency search space is wide enough, the Parallel Code-phase Search can be further optimized by using shifting in the frequency domain, providing then a smaller mean acquisition time than the Parallel Frequency Search architecture (3.21 s against 3.62 s for 10 satellites). However, if GNSS signals with higher code frequencies are considered, the Doppler effect on codes with moderate or long integration times is fatal to the PFS architecture, which loses much of its interest since the FFT will only be able to search a few bins. In this case, the PCS will then provide better performance.

The comparison carried out in this article is based on the three classical well-known architectures where no special techniques are used. Those who wish to compare a particular version of an architecture (like is sometimes the case with the PCS in order to reduce the size of the FFT in exchange for reduced SNR) can do so easily since all the formulas as well as an example are provided in Appendices 1 and 2, respectively.

Appendix 1

In this section, we provide details of the resource usage estimates of the different functions, and combine them to estimate the resources of the complete architectures. The estimates are based on FPGAs from Altera. They consist of two different basis blocks, a Logical Element (LE) used in the Cyclone and old Stratix series, and an Adaptive Logic Module (ALM) used in recent Cyclone, Arria and Stratix series. An LE consists of one register and a 4-input LUT, whereas an ALM consists of two registers and two 4-input ALUTs (Adaptive LUTs). The following conversion ratio is stated by Altera : 1 ALM = 2.5 LEs, but it is not an exact formula that works all the time. According to our experience, the ratio tends to be 1 ALM = 2 LEs most of the time, which is the ratio of their registers. In order to obtain the most accurate results possible, the estimate of the different functions is done for both basis blocks.

The formulas provided here are empirical and obtained by analysis and verifications of compilation. Implementation inside a complete system would affect the real resources usage as well as the different optimizations performed during compilation (e.g. maximizing the clock frequency, or minimizing the area). Note that the most important estimates are those of duplicated functions (or those linked to duplication), namely the code mixer, the coherent accumulator, the multiplexer, the FFT, and the coherent and non-coherent memory-based accumulators, which are not the most difficult functions to estimate.

In the following equations, we use the following notations :

- N_X : An integer (e.g. point, chip, sample)
- L_X : Resource in terms of logical blocks
- M_X : Resource in terms of memory blocks
- D_X : Resource in terms of DSP blocks
- R_X : Resolution of a signal (bits)
- R_0 : Resolution after the carrier mixer (bits)
- R_C : Resolution after the coherent accumulator (bits)
- R_{FFT} : Resolution after the FFT (bits)
- R_{NC} : Resolution after the non-coherent accumulator (bits)

Note that the resolution of the output of an accumulator performing K accumulations is

$$R_{OutAcc} = R_{InAcc} + \lceil \log_2(K) \rceil \tag{20}$$

where R_{InAcc} is the resolution of the input signal in bits.

A. Resource Estimates of Blocks

1) Carrier Generator

The carrier generator is composed of an NCO and a mapping to generate sine and cosine waveforms. Taking a 32-bit counter, the NCO thus needs 64 bits (32 bits for the counter increment and 32 bits for the counter value). The mapping is a very simple combinatorial function, and requires nothing or just a few elements and is neglected here. The resources can thus be estimated as

$$\begin{aligned} L_{CaGe} &= 64 \text{ LEs} \\ &= 32 \text{ ALMs.} \end{aligned} \tag{21}$$

2) Carrier Mixer

The carrier mixer is composed of four mixers, one adder and one subtractor (if the input signal is real, only two mixers are required). The resolution after the adder and the subtracter is denoted as R_0, and the resolution at the output of the mixers is then $R_0 - 1$. This directly provides the number of LEs, but the number of ALMs is the same as the number of LEs for the mixers. The resources can thus be estimated as

$$\begin{aligned} L_{CaMi} &= 2R_0 + 4\left(R_0 - 1\right) = 6R_0 - 4 \text{ LEs} \\ &= R_0 + 4\left(R_0 - 1\right) = 5R_0 - 4 \text{ ALMs.} \end{aligned} \tag{22}$$

3) Code Generator

Like for the carrier generator, the code generator is composed of an NCO, plus a memory where the code is stored. This necessitates more logic elements to access the memory. If several shifted versions have to be generated (denoted as N_B in the formula since it corresponds to the number of branches in the architectures), this will require one register per replica. The resources can thus be estimated as

$$\begin{aligned} L_{CoGe} &= N_B + 64 + \log_2\left(N_{chip}\right) \text{ LEs} \\ &= \frac{N_B}{2} + 32 + \log_4\left(N_{chip}\right) \text{ ALMs.} \end{aligned} \tag{23}$$

Regarding the memory, we need as many bits as there are chips in the code (insofar as the code has only two levels) :

$$M_{CoGe} = N_{chip} \text{ bits}. \tag{24}$$

4) Code Mixer & Coherent Accumulator (logic-based)

These blocks are used only in the SS and PFS architectures. The code mixer consists in inverting the I and Q signals according to the value of the code, and it is followed by a complex accumulator. In order to optimize the implementation, both blocks can be combined to build an accumulator that adds or subtracts the input value according to the value of the code. This optimization works well with ALM-based FPGAs since it does not require more resources than the accumulator alone. However, for LE-based FPGAs, it uses slightly more resources than the two blocks apart, so in this case it is better to keep them separate. The accumulators also need a signal to start/restart the integration. The resources can thus be estimated as

$$\begin{aligned} L_{CoMi} &= 2R_0 \text{ LEs} \\ L_{CoAcc} &= 2R_C + 1 \text{ LEs} \\ L_{CoMiAcc} &= R_C + 0.5 \text{ ALMs.} \end{aligned} \tag{25}$$

5) Multiplexer

This block is used only in the SS and PFS architectures. The multiplexer is fully combinatorial; consequently its resource estimate has been evaluated empirically. The resources with N inputs of R_C bits can be estimated as

$$\begin{aligned} L_{MUX} &= \frac{N R_C}{0.75} \text{ LEs} \\ &= \frac{N R_C}{1.5} \text{ ALMs.} \end{aligned} \tag{26}$$

6) Magnitude Calculator

There are many possible algorithms for the computation of the magnitude. Here we consider the Robertson approximation. The estimate has been obtained empirically, and it is a piecewise linear function depending on the resolution of the input, R.

$$\begin{aligned} L_{Mag} &= 3R + f_1\left(R\right) \text{ LEs} \\ &= 1.5R + f_2\left(R\right) \text{ ALMs} \end{aligned} \tag{27}$$

with

$$f_1\left(R\right) = \begin{cases} 33, & \text{for } 10 \le R \le 16 \\ 67, & \text{for } 17 \le R \le 32 \\ 99, & \text{for } 33 \le R \le 40 \end{cases} \tag{28}$$

$$f_2\left(R\right) = \begin{cases} 22, & \text{for } 12 \le R \le 20 \\ 42, & \text{for } 21 \le R \le 40 \end{cases}. \tag{29}$$

7) Complex Multiplier

This function is used only in the PCS architecture. It consists of four multiplications and addition/subtraction. This is done by a DSP block. If the resolution of the input is less than or equal to the basis DSP elements (18 bits for Altera FPGAs), it requires four blocks, otherwise it requires sixteen blocks.

$$D_{CMul} = \begin{cases} 4, & \text{for } 18 \le R \\ 16, & \text{for } R > 18 \end{cases} \text{ DSP elements.} \tag{30}$$

8) Ping-Pong Buffer

This function is used only in the PFS architecture. It is composed of two memories. The number of addresses of each buffer corresponds to the number of branches multiplied by the number of signal points in the FFT, and four address buses are needed to write and read both buffers.

$$\begin{aligned} L_{PPB} &= 4\log_2\left(N_B N_{FFT,S}\right) \text{ LEs} \\ &= 2\log_2\left(N_B N_{FFT,S}\right) \text{ ALMs.} \end{aligned} \tag{31}$$

The number of bits needed corresponds to the number of addresses multiplied by four times the resolution of the input signal (I & Q path, in two memories to avoid the overwriting of data not yet read).

$$M_{PPB} = 4 N_B N_{FFT,S} R_C \text{ bits.} \tag{32}$$

9) Coherent Accumulator (memory-based)

This function is used only in the PCS architecture. It consists for each signal path (I and Q) of a memory, an adder and a 2-input multiplexer. We also count the read and write address buses needed to access the memory.

$$\begin{aligned} L_{CoAcc} &= 2\left(R_C + R_C\right) + 2\log_2\left(N_{FFT}\right) \\ &= 4R_C + 2\log_2\left(N_{FFT}\right)\text{LEs} \\ &= 2\left(\frac{R_C}{2} + \frac{R_C}{2}\right) + \log_2\left(N_{FFT}\right) \\ &= 2R_C + \log_2\left(N_{FFT}\right)\ \text{ALMs}. \end{aligned} \tag{33}$$

The number of bits corresponds to the number of points in the FFT multiplied by twice the resolution of the input signal (I & Q path).

$$M_{CoAcc} = 2N_{FFT}\,R_C\ \text{bits}. \tag{34}$$

10) Non-Coherent Accumulator (memory-based)

This is the same block as the coherent accumulator except that there is now only one input signal instead of two.

$$\begin{aligned} L_{NoCoAcc} &= 2R_{NC} + 2\log_2\left(N_@\right)\ \text{LEs} \\ &= R_{NC} + \log_2\left(N_@\right)\ \text{ALMs} \end{aligned} \tag{35}$$

where $N_@$ is the number of addresses and is defined as

$$N_@ = \begin{cases} N_B, & \text{for the SS architecture} \\ N_B\,N_{FT}, & \text{for the PFS architecture} \\ N_{FFT}, & \text{for the PCS architecture} \end{cases}. \tag{36}$$

The number of bits corresponds to the number of addresses multiplied by the resolution of the input signal.

$$M_{NoCoAcc} = N_@\,R_{NC}\ \text{bits} \tag{37}$$

11) FFT

The resource usage of the FFT depends on a lot of parameters, and is estimated with the tools provided by the manufacturer or after a compilation.

B. Resource Estimates of Architectures

1) Serial Search

The functions present in the SS architecture and their sizes in terms of logical blocks are summarized in Table VI for N_B branches.

TABLE VI
LOGICAL RESOURCE ESTIMATES OF SS ARCHITECTURE

Function	Number of LEs	Number of ALMs
Carrier Generator	64	32
Carrier Mixer	6 R_0 – 4	5 R_0 – 4
Code Generator	N_B + 64 + $\log_2$ (N_{chip})	N_B/2 + 32 + $\log_4$ (N_{chip})
Code Mixers	2 N_B R_0	N_B (R_C + 0.5)
Coherent Accumulators	N_B (2 R_C + 1)	
Multiplexer	N_B R_C / 0.75	N_B R_C / 1.5
Magnitude Calculator	3 R_C + f_1(R_C)	1.5 R_C + f_2(R_C)
Non-Coherent Accumulator	2 R_{NC} + 2 $\log_2$ (N_B)	R_{NC} + $\log_2$ (N_B)

The total number of logical blocks of the SS architecture is obtained by summing all the elements of Table VI and is given by (38) for LE-based FPGAs and (39) for ALM-based FPGAs.

$$\begin{aligned} L_{SS,LE} &= N_B\left(2R_0 + \frac{10}{3}R_C + 2\right) + 2\log_2\left(N_B\right) \\ &\quad + 6R_0 + 3R_C + f_1\left(R_C\right) + 2R_{NC} \\ &\quad + \log_2\left(N_{chip}\right) + 124 \end{aligned} \tag{38}$$

$$\begin{aligned} L_{SS,ALM} &= N_B\left(\frac{5}{3}R_C + 1\right) + \log_2\left(N_B\right) \\ &\quad + 5R_0 + 1.5R_C + f_2\left(R_C\right) + R_{NC} \\ &\quad + \log_4\left(N_{chip}\right) + 60 \end{aligned} \tag{39}$$

The memory blocks are used only by the non-coherent accumulator, and the total number of bits is

$$M_{SS} = N_B\ R_{NC}. \tag{40}$$

2) Parallel Frequency Search

The functions present in the PFS architecture and their sizes in terms of logical blocks are summarized in Table VII for N_B branches.

TABLE VII
LOGICAL RESOURCE ESTIMATES OF PFS ARCHITECTURE

Function	Number of LEs	Number of ALMs
Carrier Generator	64	32
Carrier Mixer	6 R_0 – 4	5 R_0 – 4
Code Generator	N_B + 64 + $\log_2$ (N_{chip})	N_B/2 + 32 + $\log_4$ (N_{chip})
Code Mixers	2 N_B R_0	N_B (R_C + 0.5)
Coherent Accumulators	N_B (2 R_C + 1)	
Multiplexer	N_B R_C / 0.75	N_B R_C / 1.5
Ping-Pong Buffer	4 $\log_2$ (N_B $N_{FFT,S}$)	2 $\log_2$ (N_B $N_{FFT,S}$)
FFT	$L_{FFT,LE}$	$L_{FFT,ALM}$
Magnitude Calculator	3 R_{FFT} + f_1(R_{FFT})	1.5 R_{FFT} + f_2(R_{FFT})
Non-Coherent Accumulator	2 R_{NC} + 2 $\log_2$ (N_B N_{FT})	R_{NC} + $\log_2$ (N_B N_{FT})

The total number of logical blocks of the PFS architecture is obtained by summing all the elements of Table VII and is given by (41) for LE-based FPGAs and (42) for ALM-based FPGAs.

$$L_{PFS,LE} = N_B\left(2R_0 + \frac{10}{3}R_C + 2\right) + 6\log_2\left(N_B\right) + L_{FFT,LE} + 4\log_2\left(N_{FFT,S}\right) + 2\log_2\left(N_{FT}\right) + 6R_0 + 3R_{FFT} + f_1\left(R_{FFT}\right) + 2R_{NC} + \log_2\left(N_{chip}\right) + 124 \quad (41)$$

$$L_{PFS,ALM} = N_B\left(\frac{5}{3}R_C + 1\right) + 3\log_2\left(N_B\right) + L_{FFT,ALM} + 2\log_2\left(N_{FFT,S}\right) + \log_2\left(N_{FT}\right) + 5R_0 + 1.5R_{FFT} + f_2\left(R_{FFT}\right) + R_{NC} + \log_4\left(N_{chip}\right) + 60 \quad (42)$$

The functions present in the PFS architecture and their sizes in terms of memory blocks are summarized in Table VIII for N_B branches.

TABLE VIII
MEMORY RESOURCE ESTIMATES OF PFS ARCHITECTURE

Function	Number of bits
Code Generator	N_{chip}
Ping-Pong Buffer	$4\ N_B\ N_{FFT,S}\ R_C$
FFT	M_{FFT}
Non-Coherent Accumulator	$N_B\ N_{FT}\ R_{NC}$

The total number of bits of the PFS architecture is obtained by summing all the elements of Table VIII and is given by (43).

$$M_{PFS} = N_B\left(4N_{FFT,S}\ R_C + N_{FT}\ R_{NC}\right) + M_{FFT} + N_{chip}\ \text{bits} \quad (43)$$

3) Parallel Code-phase Search

The functions present in the PCS architecture and their sizes in terms of logical blocks are summarized in Table IX for N_B branches.

TABLE IX
LOGIC RESOURCE ESTIMATES OF PCS ARCHITECTURE

Function	Number of LEs	Number of ALMs
Carrier Generator	$64\ N_B$	$32\ N_B$
Carrier Mixers	$N_B\ (6\ R_0 - 4)$	$N_B\ (5\ R_0 - 4)$
Code Generator	$65 + \log_2(N_{chip})$	$32.5 + \log_4(N_{chip})$
FFTs	$(N_B + 1)\ L_{FFT,LE}$	$(N_B + 1)\ L_{FFT,ALM}$
Complex Multipliers	0	0
IFFTs	$N_B\ L_{IFFT,LE}$	$N_B\ L_{IFFT,ALM}$
Coherent Accumulators	$4\ N_B\ R_C + 2\ \log_2(N_{FFT})$	$2\ N_B\ R_C + \log_2(N_{FFT})$
Magnitude Calculators	$N_B\ [3\ R_C + f_1(R_C)]$	$N_B\ [1.5\ R_C + f_2(R_C)]$
Non-Coherent Accumulators	$N_B\ 2\ R_{NC} + 2\ \log_2(N_{FFT})$	$N_B\ R_{NC} + \log_2(N_{FFT})$

The total number of logical blocks of the PCS architecture is obtained by summing all the elements of Table IX and is given by (44) for LE-based FPGAs and (45) for ALM-based FPGAs.

$$L_{PCS,LE} = N_B\left[L_{FFT,LE} + L_{IFFT,LE} + 7R_C + 2R_{NC} + f_1\left(R_C\right) + 6R_0 + 60\right] + L_{FFT,LE} + 4\log_2\left(N_{FFT}\right) + \log_2\left(N_{chip}\right) + 65 \quad (44)$$

$$L_{PCS,ALM} = N_B\left[L_{FFT,ALM} + L_{IFFT,ALM} + 3.5R_C + R_{NC} + f_2\left(R_C\right) + 5R_0 + 28\right] + L_{FFT,ALM} + 2\log_2\left(N_{FFT}\right) + \log_4\left(N_{chip}\right) + 32.5 \quad (45)$$

The functions present in the PCS architecture and their sizes in terms of memory blocks are summarized in Table X for N_B branches.

TABLE X
MEMORY RESOURCE ESTIMATES OF PCS ARCHITECTURE

Function	Number of bits
Code Generator	N_{chip}
FFTs	$(N_B + 1)\ M_{FFT}$
IFFTs	$N_B\ M_{IFFT}$
Coherent Accumulators	$2\ N_B\ N_{FFT}\ R_C$
Non-Coherent Accumulators	$N_B\ N_{FFT}\ R_{NC}$

The total number of bits of the PCS architecture is obtained by summing all the elements of Table X and is given by (46).

$$M_{PCS} = N_B\left[M_{FFT} + M_{IFFT} + N_{FFT}\left(2R_C + R_{NC}\right)\right] + M_{FFT} + N_{chip}\ \text{bits} \quad (46)$$

The functions present in the PCS architecture and their sizes in terms of DSP blocks are summarized in Table XI for N_B branches.

TABLE XI
DSP RESOURCE ESTIMATES OF PCS ARCHITECTURE

Function	DSP 18-bit elements
FFTs	$(N_B + 1)\ D_{FFT}$
Complex Multipliers	$N_B\ D_{CMUL}$

IFFTs	$N_B\, D_{IFFT}$

The total number of DSP elements of the PCS architecture is obtained by summing all the elements of Table XI and is given by (47).

$$D_{PCS} = N_B \left(D_{FFT} + D_{IFFT} + D_{CMUL} \right) + D_{FFT} \quad (47)$$

APPENDIX 2

In this section, we provide the details of the application example used in Section IV with the acquisition parameters summarized in Table IV. First, the target FPGA is presented, then for each architecture all values (number of accumulations, resolution of signals and size of FFTs) are specified, and the formulas of Appendix 1 are used to determine the number of branches that can be implemented. The results are summarized in Table V.

A. Application with a Low-Cost FPGA Series : Cyclone III

The target chip considered is the EP3C120 with the following resources :

- 119 088 LEs
- 432 blocks of 9216 bits (9216 bits = 1 M9K)
- 288 18-bit multipliers

Taking into account that only 85 % of the FPGA logical blocks can be used, and considering the other functions in the FPGA such as the tracking channels, the management, the processor, etc. (evaluated to 20 000 LEs according to our experience), we arrive at about 80 000 LEs available for the acquisition channel. Note that the assumptions made here impact the absolute results (i.e. performances), but not the comparison between the different architectures (i.e. the ranking).

1) Serial Search

The characteristics are summarized in Table XII.

TABLE XII
SS ARCHITECTURE PARAMETERS

Parameter	Value
R_0	5 bits
N_C	40 960
R_C	21 bits
N_{NC}	40
R_{NC}	27 bits

Using (38), we obtain

$$80\,000 \geq 82\, N_B + 2\log_2\left(N_B\right) + 348 \quad (48)$$

from which we deduce that the maximum number of branches implementable is $N_B = 971$.

2) Parallel Frequency Search

The characteristics are summarized in Table XIII. For an FFT of this size, the implementation that uses the natural and bit-reversed order requires fewer logic and memory resources (but more DSP resources) than the implementation that uses only the natural order; consequently the evaluation is made for the first implementation (DSP resources are not critical with this architecture).

TABLE XIII
PFS ARCHITECTURE PARAMETERS WITH CYCLONE III FPGA

Parameter	Value
R_0	5 bits
N_C / f_S	625 µs
N_C	2560
R_C	17 bits
$N_{FFT,S}$	16
N_{FFT}	32
R_{FFT}	18 bits
N_{NC}	40
R_{NC}	24 bits
L_{FFT}	4359 LEs
M_{FFT}	6 M9Ks = 55 296 bits

Using (41), we obtain

$$80\,000 \geq \frac{206}{3} N_B + 6\log_2\left(N_B\right) + 4716 \quad (49)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is $N_B = 1095$. Using (43), we obtain

$$425 \times 9216 \geq 1352\, N_B \quad (50)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is $N_B = 2897$. Consequently the limitation comes from the logical blocks.

3) Parallel Code-phase Search

The characteristics are summarized in Table XIV.

TABLE XIV
PCS SEARCH ARCHITECTURE PARAMETERS WITH CYCLONE III FPGA

Parameter		Value
R_0		5 bits
N_{FFT}		4096
R_{FFT}		18 bits
N_C		10
R_C		22 bits
N_{NC}		40
R_{NC}		28 bits
FFT with input and output in natural order	L_{FFT} , L_{IFFT}	7756 LEs
	M_{FFT} , M_{IFFT}	76 M9Ks = 700 416 bits
	D_{FFT} , D_{IFFT}	24 DSP elements
FFT with input	L_{FFT}	9962 LEs

in natural order and output in bit-reversed order	M_{FFT}	37 M9Ks = 340 992 bits
	D_{FFT}	40 DSP elements
IFFT with input in bit-reversed order and output in natural order	L_{IFFT}	10 149 LEs
	M_{IFFT}	48 M9Ks = 442 368 bits
	D_{IFFT}	40 DSP elements

Let's consider first the FFTs with only the natural order. Using (44), we obtain

$$80\,000 \geq 15\,879 N_B + 7879 \quad (51)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is $N_B = 4$. Using (46), we obtain

$$355 \times 9216 \geq 1\,695\,744 N_B \quad (52)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is $N_B = 1$. Using (47), we obtain

$$288 \geq 52 N_B + 24 \quad (53)$$

from which we deduce that the maximum number of branches implementable due to the DSP resources is $N_B = 5$. Consequently the limitation comes from the memory resources. Now, let's consider the FFTs with the natural and bit-reversed order. Using (44), we obtain

$$80\,000 \geq 20\,478 N_B + 10\,085 \quad (54)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is $N_B = 3$. Using (46), we obtain

$$394 \times 9216 \geq 1\,078\,272 N_B \quad (55)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is $N_B = 3$. Using (47), we obtain

$$288 \geq 84 N_B + 40 \quad (56)$$

from which we deduce that the maximum number of branches implementable due to the DSP resources is $N_B = 2$. In this case, the limitation comes from the DSP blocks. We note that this implementation of the FFT is better since two branches can be implemented instead of one.

B. Application with a High-End FPGA Series : Stratix III

Now, the target chip considered is the EP3SE260 with the following resources :

- 135 200ALMs
- 864 blocks of 9216 bits (M9K)
- 48 blocks of 147 456 bits (M144K = 16 M9K)
- 768 18-bit multipliers

The remark made before regarding the space in the FPGA remains valid here, and we consider that 105 000 ALMs are available for the acquisition channel. The number of accumulations and the resolution of signals are identical to those already indicated and are not repeated here.

1) Serial Search

Using (39), we obtain

$$105\,000 \geq 36 N_B + \log_2(N_B) + 191 \quad (57)$$

from which we deduce that the maximum number of branches implementable is $N_B = 2911$.

2) Parallel Frequency Search

The characteristics are summarized in Table XV.

TABLE XV
PFS ARCHITECTURE PARAMETERS WITH STRATIX III FPGA

Parameter	Value
L_{FFT}	1790 ALMs
M_{FFT}	2 M9Ks = 18 432 bits

Using (42), we obtain

$$105\,000 \geq \frac{88}{3} N_B + 3\log_2(N_B) + 1970 \quad (58)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is $N_B = 3511$. Using (43), we obtain

$$1629 \times 9216 \geq 1352 N_B \quad (59)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is $N_B = 11104$. Consequently the limitation comes from the logic resources. Due to the limitation in the multiplexing described in Section III.D, it is necessary to use three multiplexer chains. Taking this into account, there are 3385 branches that can be implemented.

3) Parallel Code-phase Search

The characteristics are summarized in Table XVI.

TABLE XVI
PCS ARCHITECTURE PARAMETERS WITH STRATIX III FPGA

Parameter		Value
FFT with input and output in natural order	L_{FFT} , L_{IFFT}	3806 ALMs
	M_{FFT} , M_{IFFT}	76 M9Ks = 700 416 bits
	D_{FFT} , D_{IFFT}	24 DSP elements
FFT with input in natural order and output in bit-reversed order	L_{FFT}	5083 ALMs
	M_{FFT}	31 M9Ks = 285 696 bits
	D_{FFT}	40 DSP elements
IFFT with input in bit-reversed order and output in natural order	L_{IFFT}	5146 ALMs
	M_{IFFT}	42 M9Ks = 387 072 bits
	D_{IFFT}	40 DSP elements

Let's consider first the FFTs with only the natural order. Using (45), we obtain

$$105\,000 \geq 7812 N_B + 3868 \quad (60)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is $N_B = 12$. Using

(46), we obtain

$$1555 \times 9216 \geq 1695744\, N_B \quad (61)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is N_B = 8. Using (47), we obtain

$$768 \geq 52\, N_B + 24 \quad (62)$$

from which we deduce that the maximum number of branches implementable due to the DSP resources is N_B = 14. Consequently the limitation comes from the memory resources. Now, let's consider the FFTs with the natural and bit-reversed order. Using (45), we obtain

$$105000 \geq 10429\, N_B + 5145 \quad (63)$$

from which we deduce that the maximum number of branches implementable due to the logic resources is N_B = 9. Using (46), we obtain

$$1600 \times 9216 \geq 967680\, N_B \quad (64)$$

from which we deduce that the maximum number of branches implementable due to the memory resources is N_B = 15. Using (47), we obtain

$$768 \geq 84\, N_B + 40 \quad (65)$$

from which we deduce that the maximum number of branches implementable due to the DSP resources is N_B = 8. In this case, the limitation comes from the DSP blocks. We note that both implementations of the FFT provide equivalent performance; the limitation may come from the memory or the DSP resources.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their judicious comments which helped to greatly improve the quality of this article.

REFERENCES

[1] A.J. Viterbi, "CDMA: Principles of Spread Spectrum Communication", Prentice Hall, 1995.

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

[3] L. Lo Presti, B. Motella, "The math of ambiguity: what is the acquisition ambiguity function and how is it expressed mathematically? ", Inside GNSS, vol. 5, no. 4, pp. 20 – 28, June 2010.

[4] D. Borio, C. O'Driscoll, G. Lachapelle, "Coherent, Noncoherent, and Differentially Coherent Combining Techniques for Acquisition of New Composite GNSS Signals", IEEE Transactions on Aerospace and Electronic Systems, vol. 45, no. 3, pp. 1227 – 1240, July 2009.

[5] A. El-Rabbany, "Introduction to GPS: The Global Positioning System", Artech House, 2006.

[6] E. Kaplan, C. Hegarty, "Understanding GPS: Principles and Applications", Artech House, 2006.

[7] K. Borre, D. Akos, N. Bertelsen, P. Rinder, S.K. Jensen, "A Software-Defined GPS and Galileo Receiver: A Single Frequency Approach", Birkhäuser Boston, 2006.

[8] S.M. Spangenberg, G.J.R. Povey, "Code acquisition for LEO satellite mobile communication using a serial-parallel correlator with FFT for Doppler estimation", CSDSP, Sheffield, UK, April 1998.

[9] H. Mathis, P. Flammant, A. Thiel, "An analytic way to optimize the detector of a post-c orrelation FFT acquisition algorithm", ION GPS/GNSS 2003, Portland, Oregon, USA, Sept. 2003.

[10] C. Botteron, G. Walchli, G. Zamuner, M. Frei, D. Manetti, F. Chastellain, P-A. Farine, P. Brault, "A Flexible Galileo L1 Receiver Platform for the Validation of Low Power and Rapid Acquisition Schemes", ION GNSS 2006, Forth Worth, Texas, USA, Sept. 2006.

[11] R. Lyons, "Reducing FFT Scalloping Loss Errors Without Multiplication", IEEE Signal Processing Magazine, vol. 28, no. 2, pp. 112 – 116, March 2011.

[12] U. Cheng, W.J. Hurd, J.I. 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, Feb. 1990.

[13] D.J.R. van Nee, A.J.R.M. Coenen, "New Fast GPS Code-Acquisition Technique using FFT", Electronics Letters, vol. 27, no. 2, pp. 158 – 160. Jan. 1991.

[14] S.W. Smith, "Digital Signal Processing: A Practical Guide for Engineers and Scientists", Newnes Press, 2002.

[15] L. Kurz, G. Kappen, T. Coenen, T.G. Noll, "Comparison of Massive-Parallel and FFT Based Acquisition Architectures for GNSS-Receivers", ION GNSS 2010, Portland, Oregon, USA, Sept. 2010.

[16] J. Leclère, C. Botteron, 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", IEEE/ION PLANS 2010, Palm Springs, California, USA, May 2010.

[17] R.G. Lyons, "Understanding Digital Signal Processing", Prentice Hall, 2010.

[18] S.U. Qaisar, N.C. Shivaramaiah, A.G. Dempster, C. Rizos, "Filtering IF Samples to Reduce the Computational Load of Frequency Domain Acquisition in GNSS Receivers", ION GNSS 2008, Savannah, Georgia, USA, Sept. 2008.

[19] J.A. Starzyk, Z. Zhu, "Averaging correlation for C/A code acquisition and tracking in frequency domain", IEEE 2001 Midwest Symposium on Circuits and Systems, Dayton, Ohio, USA, Aug. 2001.

[20] N.C. Shivaramaiah, A.G. Dempster, C. Rizos, "Application of Mixed-radix FFT Algorithms in Multi-band GNSS Signal Acquisition Engines", Journal of Global Positioning Systems, vol. 8, no. 2, pp. 174 – 186, 2009.

[21] L. Lo Presti, Z. Xuefen, M. Fantino, P. Mulassano, "GNSS Signal Acquisition in the Presence of Sign Transition", IEEE Journal of Selected Topics in Signal Processing, vol. 3, no. 4, pp. 557 – 570, Aug. 2009.

[22] M.L. Psiaki, "Block Acquisition of Weak GPS Signals in a Software Receiver", ION GPS 2001, Salt Lake City, Utah, USA, Sept. 2001.

[23] J.B. Lozow, "Analysis of Direct P(Y)-Code Acquisition", Journal of the Institute of Navigation, vol. 44, no. 1, pp. 89 – 98, Spring 1997.

[24] J.K. Holmes, C.C. Chen., "Acquisition Time Performance of PN Spread-Spectrum Systems", IEEE Transactions on Communication, vol. 25, no. 8, pp. 778 – 784, Aug. 1977.

[25] D. Borio, L. Camoriano, L. Lo Presti, "Impact of GPS Acquisition Strategy on Decision Probabilities", IEEE Transactions on Aerospace and Electronic Systems, vol. 44, no. 3, pp. 996 – 1011, July 2008.

[26] N.C. Shivaramaiah, H.S. Jamadagni, M. Srikantaiah, V. Chikkabbaiah, "Software-Aided Sequential Multi-tap Correlator for Fast Acquisition", ION GNSS 2004, Long Beach, California, USA, Sept. 2004.

[27] D. Akopian, "Fast FFT based GPS satellite acquisition method", IEE Proceedings - Radar, Sonar and Navigation, vol. 152, no. 4, pp. 277 – 286, Aug. 2005.

[28] Altera, "Designing and Using FPGAs for Double-Precision floating-Point Math", White Paper, 2007.

Jérôme Leclère received the Master Degree in Electronics and Signal Processing from the ENSEEIHT, Toulouse, France, in 2008. He is currently performing his Ph.D. thesis in the GNSS field at EPFL in the Electronics and Signal Processing Laboratory, focusing his researches in the acquisition and high sensitivity areas, with application to hardware receivers, especially using FPGAs. He participated to the development of an FPGA-based high sensitivity GPS L1 C/A receiver.

Dr. Cyril Botteron (M'99, SM06) received the Electronics Engineering degree from the University of Applied Sciences in Le Locle, Switzerland, in 1991, and the PhD degree with specialization in wireless communications from the University of Calgary, Canada, in 2003.
From 1991 to 2000, he worked as design engineer for different companies in the USA, in Switzerland, and in Canada. From 2003 to 2008, he was a lecturer and team leader at University of Neuchâtel, Switzerland, leading research in radio frequency (RF) microelectronic circuits and wireless system designs. Since 2009, he is working at École Polytechnique Fédérale de Lausanne, Switzerland, leading, managing, and coaching the research and project activities of the GNSS and UWB subgroups in the Electronics and Signal Processing Laboratory. His current research interests comprise the development of low-power RF integrated circuits and advanced signal processing techniques for ultra-low-power communications and positioning applications (using GNSS, INS, UWB, etc.). He is the author or coauthor of more than 50 articles in international journals and conference proceedings and 4 patent families.
Dr. Botteron has received many awards, including in 2001 a postgraduate scholarship and a graduate fellowship from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE), respectively.

Prof. Pierre-André Farine (M'85) was born in La Chaux-de-Fonds, Switzerland, in 1953. He received the B.Sc. degree in microtechnology from École Technique Supérieure (ETS), Le Locle, Switzerland, in 1974 and the M.Sc. and Ph.D. degrees in Microtechnology from the University of Neuchâtel, Switzerland, in 1978 and 1984, respectively.
For 17 years, he was with the Swiss watch industries (Swatch Group) in the development of high-tech products, like pagers and watches with integrated sensors such as pressure, compass, altimeter, and temperature. He was also involved in prototype developments for watches including GPS and cellular phones subsystems. Between 2002 and 2009, he was professor with the Institute of Microtechnology, University of Neuchâtel, where he was leading the Electronics and Signal Processing Laboratory, 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. Since 2009, he is full professor with the École Polytechnique Fédérale de Lausanne (EPFL), where he is leading the Electronics and Signal Processing Laboratory EPFL ESPLAB. He is the author or coauthor of more than 100 publications in conference and technical journals and more 50 patent families. Prof. Farine has served as a member of program committees and the program chair for international conferences.