Presented at ION GNSS 2012, Session D3, Nashville, TN, USA, September 17-21, 2012.

Improving the Performance of the FFT-based Parallel Code-phase Search Acquisition of GNSS Signals by Decomposition of the Circular Correlation

Jérôme Leclère, Cyril Botteron, Pierre-André Farine,
Electronics and Signal Processing Laboratory (ESPLAB),
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

BIOGRAPHIES

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, focusing his researches in the acquisition and high sensitivity areas, with application to hardware receivers, especially using FPGAs.

Dr. Cyril Botteron leads the GNSS and UWB groups in the electronics and signal processing laboratory at EPFL. He received his PhD degree from the University of Calgary, Canada, in 2003. His current research interests comprise the development of low power radio frequency (RF) integrated circuits and advanced signal processing techniques for ultra-low power communications and global and local positioning applications.

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

ABSTRACT

This paper proposes alternative architectures to perform a circular correlation using the Fast Fourier Transform (FFT) by decomposing the initial circular correlation into several smaller circular correlations. The approach used is similar to the Fast Finite Impulse Response (FIR) Algorithms (FFAs). These architectures improve the performance in terms of reduced processing time or resource usage, and consequently lower the energy consumption.
The results can be applied to any system that performs circular convolution or correlation. In this paper, the application is the acquisition of Global Navigation Satellite System (GNSS) signals with the FFT-based Parallel Code-phase Search (PCS), and more precisely on the GPS L1 C/A signal, when the target considered is a Field Programmable Gate Array (FPGA).
In this context, it is for example shown that it is possible with one of the proposed architectures to reduce the logic usage by 11 %, the memory usage by 41 %, and the Digital Signal Processing (DSP) block usage by 32 %, while keeping the same processing time. With another architecture, it is shown that the processing time can be halved by increasing the logic usage by only 35 %, while reducing the memory usage and keeping the same DSP usage.
Note that the proposed approach is not based on an approximation of the traditional method, but a modified implementation providing the same result. Thus, there is no loss of sensitivity.

1. INTRODUCTION

The acquisition of GNSS signals consists mainly in three steps : 1) multiplication of the input signal with a local carrier replica to remove the offset in frequency due to the Doppler effect, 2) multiplication with a local pseudo-random noise (PRN) code replica; 3) Integration. The process has to be repeated for different carrier frequencies and code phases of the replicas until they are both aligned with the received ones. This is thus a two-dimension search (for each satellite). Together, the last two steps are equivalent to a circular correlation (due to the code repetition). As a consequence, the FFT can be used to compute it. This enables getting the correlation result for all the code-phases simultaneously. This is known as Parallel Code-phase Search acquisition.
However, the processing time can still be relatively long, because of : 1) the different carrier frequencies to test; 2) the results over dozens or hundreds of code periods that are usually accumulated to increase the sensitivity; 3) the process has to be repeated for several satellites (up to a few dozens, if there is no a priori information and several constellations are considered). Therefore, finding new methods to perform the search faster is still topical.
Within this context, we use an approach that consists in decomposing the initial circular correlation into several

smaller circular correlations. This approach is similar to the FFAs, and the idea to combine it with the FFT to perform convolution was briefly described in [1], but the algorithms proposed were not optimal and no deep study was undertaken.

Therefore, in this paper we propose and study several architectures using different numbers of FFTs that lead to different performance results. Among them, one architecture enables reducing the resources while keeping the same processing time, whereas others enable reducing the processing time, at the expense of an increase in resources.

The rest of the paper is organized as follows : In Section 2, a fast review of the acquisition of GNSS signals is given. Section 3 shows how the FFT can be used to compute a circular correlation. Section 4 shows how the processing time can be reduced by duplicating elements. Section 5 presents the FFA principle and provides different architectures to reduce the processing time or the resource usage. Section 6 discusses briefly the impact on the energy consumption. In Section 7, an application example is discussed, providing details about the FPGAs and models used for the FFT and other functions. Results are first obtained using the models presented, and then checked with a real implementation of two architectures. Section 8 summarizes some important results and concludes on the impact of the proposed architectures.

2. ACQUISITION OF GNSS SIGNALS

Signals transmitted by GNSS satellites are a combination of a carrier, one or several PRN codes specific to each satellite, and a navigation data message [2]. When reaching the GNSS receiver, the frequency of the carrier (and of the PRN code) is different for each satellite because of the Doppler effect, and the phase of the code is unknown since the distance from the satellites is unknown assuming no a priori synchronization.

After down-conversion and digitization, the GNSS receiver performs an acquisition, which consists in multiplying the signal s[n] by a carrier replica of the same frequency; then multiplying by a code replica of the same phase (and same frequency), c[n-τ], where τ is a delay; and then integrating the result over time to raise the signal out of the noise [3]. The last two steps are equivalent to a circular correlation (due to the code repetition).

The corresponding schematic is shown in Fig. 1, where it can be noted that the process repeats for different phases of the code (τ belongs to *T*) and different carrier frequencies called bins (f belong to *F*). *T* and *F* depend on the context, such as the PRN code length, the signal modulation, the carrier frequency, the integration time or a priori information. For example, considering the GPS L1 C/A signal which PRN code is 1023-chip long (corresponding to 1 ms) and the frequency range of about ± 5 kHz (for a static user, and not including the offset of the receiver oscillator [4]), the code-phase step will be typically ½ chip resulting in 2046 code bins; and the frequency step will be typically 500 Hz for a coherent integration time of 1 ms leading to 21 frequency bins [4].

The search can be parallelized in the frequency space by looking at several or all the frequency bins using an FFT : this is called Parallel Frequency Search [5][6]. The search can also be parallelized in the code space, by using the FFT to get the correlation result for all the code bins simultaneously : this is the Parallel Code-phase Search (PCS) [7]. A description of these methods and their implementation on FPGAs can be found in [8]. In the following, we concentrate on the PCS, which is described in the next section.

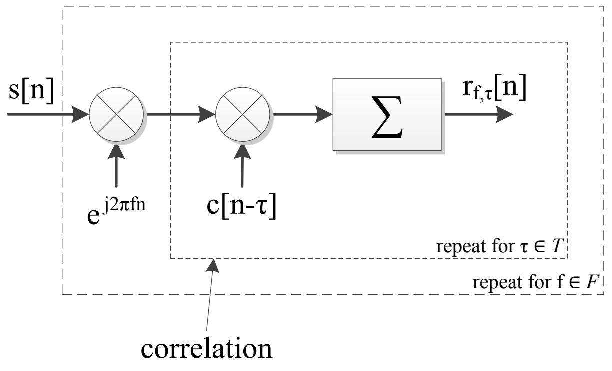

Fig. 1 : Principle of the acquisition of GNSS signals

3. CIRCULAR CORRELATION USING FFT

The circular correlation between two finite-length sequences x[n] and h[n] (corresponding to the input signal and code replica in our case, respectively), is defined by Eq. (1), where N is the number of samples in one period of the PRN code and * denotes the conjugate.

$$y[n] = \sum_{k=0}^{N-1} h^*[k]x[(k+n) \bmod N] \quad (1)$$

In z domain formulation, the correlation becomes a product of two polynomials [9].

$$Y(z) = X(z)\,H^*(1/z^*) \bmod (z^{-N}-1) \quad (2)$$

The Discrete Fourier Transform (DFT) of y[n] can be obtained by evaluating Y(z) in z = $e^{j2\pi k/N}$, and we obtain

$$\mathrm{DFT}\left[y[n]\right] = \mathrm{DFT}\left[x[n]\right]\mathrm{DFT}^*\left[h[n]\right] \quad (3)$$

Using the FFT algorithm to implement the DFT [10], the circular correlation can thus be obtained using the Inverse FFT (IFFT), as shown by Eq. (4).

$$y[n] = \mathrm{IFFT}\left[\mathrm{FFT}\left[x[n]\right]\mathrm{FFT}^*\left[h[n]\right]\right] \quad (4)$$

In the GNSS context, the code replica is real (for quadrature signals such as L5 and E5, the codes of pilot and data channels can be generated as complex or separately as real as well). Using this property and the fact that the conjugate of the FFT of a sequence is equal to the IFFT of the conjugate of the same sequence [11], Eq. (4) can be simplified to Eq. (5).

$$y[n] = \mathrm{IFFT}\left[\mathrm{FFT}\left[x[n]\right]\mathrm{IFFT}\left[h[n]\right]\right] \quad (5)$$

The circular correlation can thus be performed using one N-point FFT, and two N-point IFFTs, as shown in Fig. 2.

For the following, the architectures will be denoted as P−T−N−M, where P corresponds to ratio between the processing time of the traditional architecture provided in this section and the processing time of the architectures (neglecting the latency); T to the number of FFTs and IFFTs used; N to the transform length of the FFTs; and M

to the number of complex multipliers used. The traditional architecture can thus be denoted as 1−3−N−1.
A simplified timing diagram is shown in Fig. 3, where the different code periods are represented by different colors. Considering that an FFT has a latency of N + L cycles, L being an intrinsic latency linked to the FFT implementation, the K^th correlation result is thus available after KN + 2N + 2L cycles.

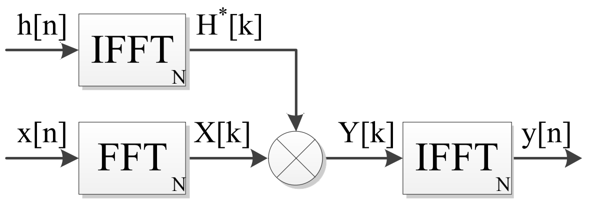

Fig. 2 : Circular correlation using FFT (architecture 1−3−N−1)

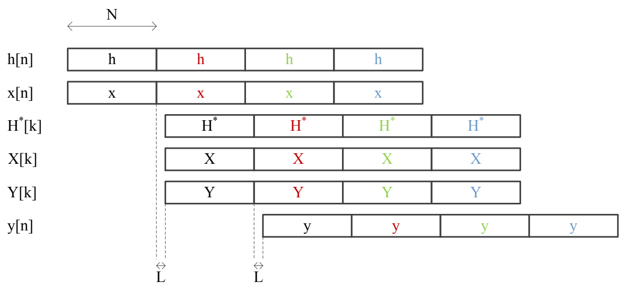

Fig. 3 : Timing diagram of the traditional architecture (1−3−N−1)

4. REDUCTION OF THE PROCESSING TIME BY DUPLICATION

4.1 Initial Approach

One of the solutions to reduce the processing time is to duplicate the architecture, as shown in Fig. 4, where the top branch can process the even periods of the code, and the bottom one the odd periods.
In this case the processing time is halved (considering an even number of correlation result), since the K^th correlation result is available after $\lceil K/2 \rceil N + 2N + 2L$ cycles (see timing diagram in Fig. 5), but the resources are doubled. In fact, the resources are slightly more than doubled due to the extra adder and the modified generation of the code replica. Indeed, this architecture requires the generation of two consecutive periods of the replica simultaneously (see Appendix A for more details on code replica generation).

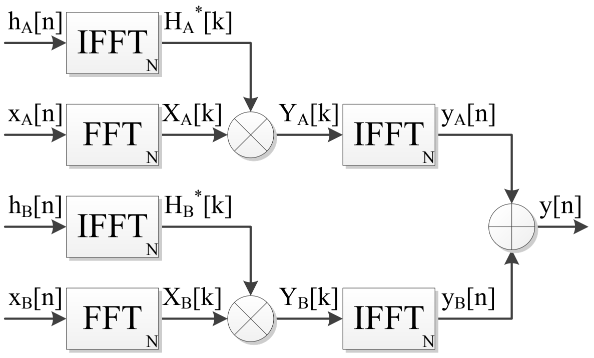

Fig. 4 : Duplication of the traditional architecture (architecture 2−6−N−2)

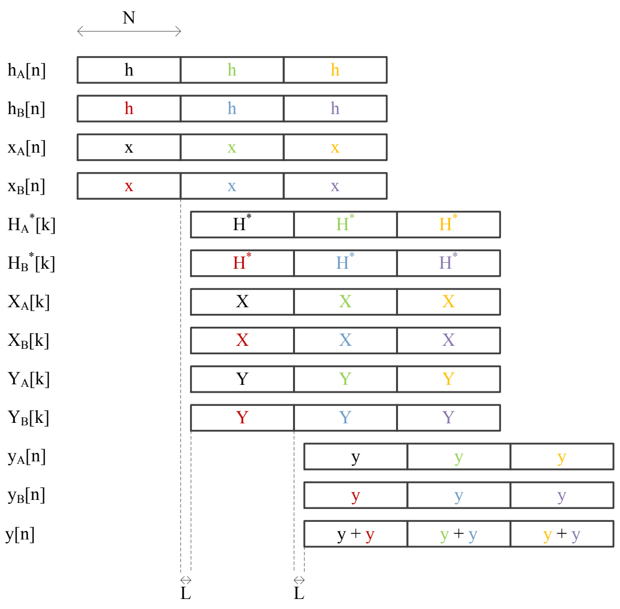

Fig. 5 : Timing diagram when the traditional architecture is duplicated (architecture 2−6−N−2)

4.2 Use of the real property of the PRN code replica h[n]

It is known that the FFTs of two real sequences can be performed using only one FFT at the expense of a recombination afterwards [12] (details on implementation are given in Appendix B). Using this principle, we can thus use one FFT for $h_A[n]$ and $h_B[n]$, and obtain the architecture shown in Fig. 6. However, this trick increases the global latency by N cycles, and the K^th correlation result is now available after $\lceil K/2 \rceil N + 3N + 2L$ cycles (see timing diagram in Fig. 7).

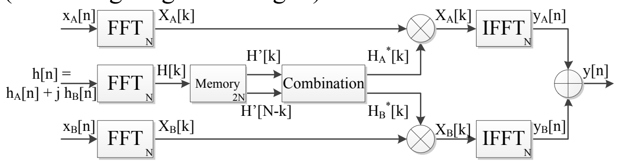

Fig. 6 : Modification of the architecture 2−6−N−2 into an architecture 2−5−N−2

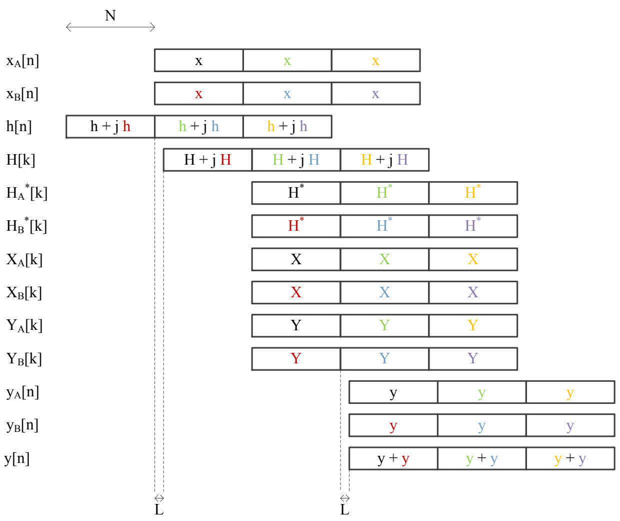

Fig. 7 : Timing diagram of the architecture 2−5−N−2

5. FAST FIR ALGORITHMS

5.1 Introduction to FFAs

Using the polyphase decomposition of filters [13], it is possible to obtain more efficient structures for FIR filters by parallelizing the processing and removing redundant computations [14][15]. A FIR filter performs a convolution, which is related to correlation; consequently we can apply the concept to the PCS.

5.2 Separation in two

5.2.1 Initial Approach

We start by using the following polyphase representation (which corresponds to a separation of the signals into even and odd samples) :

$$\begin{aligned} Y(z) &= Y_0(z^2) + z^{-1}Y_1(z^2) \\ X(z) &= X_0(z^2) + z^{-1}X_1(z^2) \qquad (6) \\ H(z^{-1}) &= H_0(z^{-2}) + z\,H_1(z^{-2})\,, \end{aligned}$$

where

$$\begin{aligned} Y_i(z) &= \sum_{n=0}^{\infty} y[2n+i]\, z^{-n} \\ X_i(z) &= \sum_{n=0}^{\infty} x[2n+i]\, z^{-n} \qquad (7) \\ H_i(z^{-1}) &= \sum_{n=0}^{\infty} h[2n+i]\, z^{n}\,, \end{aligned}$$

and i ∈ {0, 1}. Using Eq. (6), we can thus reformulate Eq. (2) as (the modulo operation is not shown) :

$$\begin{aligned} Y_0(z^2) &= H_0(z^{-2})X_0(z^2) + H_1(z^{-2})X_1(z^2) \\ Y_1(z^2) &= z^2\, H_1(z^{-2})X_0(z^2) + H_0(z^{-2})X_1(z^2) \end{aligned} \qquad (8)$$

Evaluating these relations in $z = e^{j2\pi k/N}$, we obtain

$$\begin{aligned} Y_0[k] &= H_0^*[k]X_0[k] + H_1^*[k]X_1[k] \\ Y_1[k] &= e^{j2\pi k/(N/2)}\, H_1^*[k]X_0[k] + H_0^*[k]X_1[k]\,, \end{aligned} \qquad (9)$$

where

$$\begin{aligned} Y_i[k] &= \text{FFT}\left[y[2n+i]\right] \\ X_i[k] &= \text{FFT}\left[x[2n+i]\right] \qquad (10) \\ H_i^*[k] &= \text{IFFT}\left[h[2n+i]\right]. \end{aligned}$$

and i ∈ {0, 1}. The corresponding architecture is shown in Fig. 8. The number of FFTs is doubled compared to the traditional architecture, however the transform length is halved, and there are now 5 multipliers and 2 adders.

A simplified timing diagram of this architecture is shown in Fig. 9. The Kth correlation result is available after KN/2 + N + 2L cycles.

Similarly as for the architectures using duplication, this architecture requires a modified generation of the code replica, since two consecutive samples of the replica must be generated simultaneously (see Appendix A).

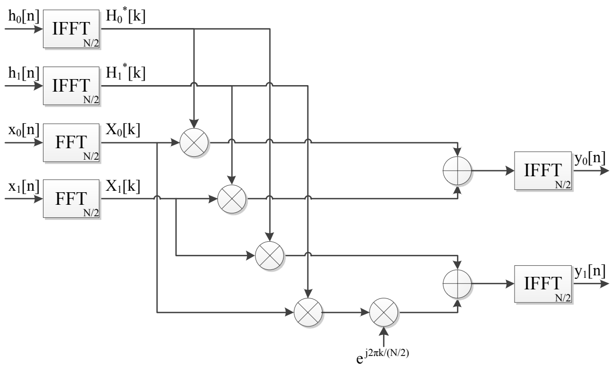

Fig. 8 : FFA-based circular correlation (architecture 2−6−N/2−5)

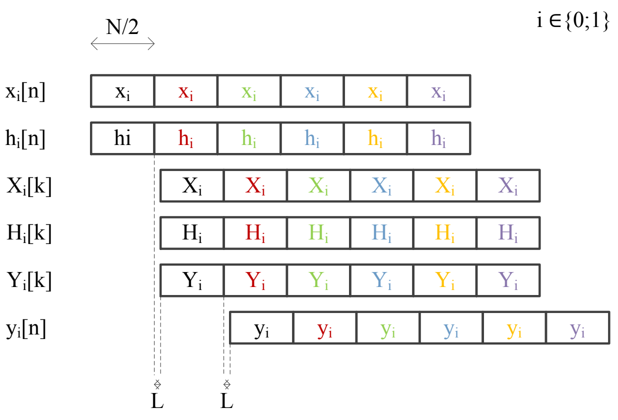

Fig. 9 : Timing diagram of the FFA-based architecture 2−6−N/2−5

5.2.2 Reduction of multipliers

It is possible to reduce the number of multipliers at the expense of extra adders, as detailed in [14] and [15]. This is interesting because multipliers cost more than adders in terms of hardware. A possible solution is given in Eq. (11).

$$\begin{aligned} Y_0[k] &= \left(H_1^*[k] - H_0^*[k]\right)X_1[k] \\ &\quad + H_0^*[k]\left(X_0[k] + X_1[k]\right) \\ Y_1[k] &= \left(e^{j2\pi k/(N/2)}H_1^*[k] - H_0^*[k]\right)X_0[k] \\ &\quad + H_0^*[k]\left(X_0[k] + X_1[k]\right) \end{aligned} \qquad (11)$$

The corresponding architecture is shown in Fig. 10.

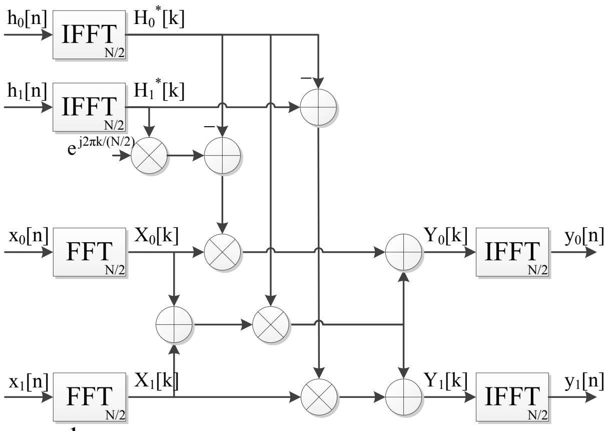

Fig. 10[1] : FFA-based circular correlation with reduced number of multipliers (architecture 2−6−N/2−4)

[1] This figure contains two typo errors : $Y_0[k]$ and $Y_1[k]$ are inverted, as well as $y_0[n]$ and $y_1[n]$.

There are now 4 multipliers, 5 adders and the generation of an exponential is required (typically using a Numerically Controlled Oscillator, or NCO). The timing diagram is not affected as compared to the architecture 2−6−N/2−5.

5.2.3 Use of the real property of the PRN code replica h[n]

As described in Section 4.2, we can use only one FFT for $h_0[n]$ and $h_1[n]$, and obtain the architecture shown in Fig. 11 with the corresponding timing diagram in Fig. 12. The K^th correlation result is now available after KN/2 + 3N/2 + 2L cycles.

Compared to the architecture 2−5−N−2, there are as many FFTs but the transform length is halved. We can thus expect that this architecture is more efficient, even if it has 2 extra multipliers and 4 extra adders.

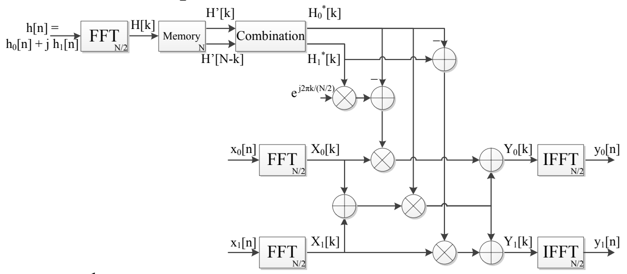

Fig. 11[1] : Modification of the architecture 2−6−N/2−4 into an architecture 2−5−N/2−4

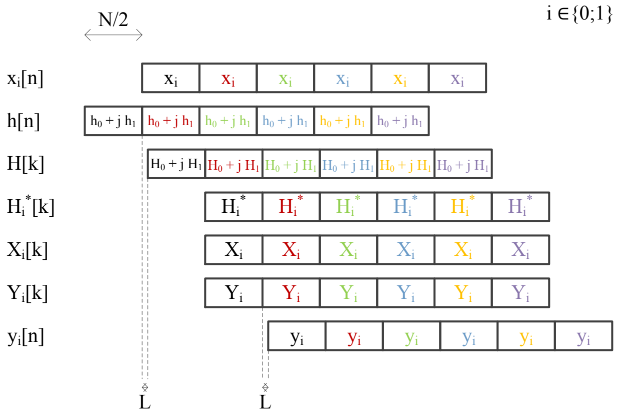

Fig. 12 : Timing diagram of the FFA-based architecture 2−5−N/2−4

5.3 Separation in P

The same principle can be applied to split the signals in 3, 4, or any value. For a splitting in P (which means a reduction of the processing time by P), when the number of multipliers and FFTs is not reduced (as architecture 2−6−N/2−5), the resulting architecture is composed of

- 3P (I)FFTs of N/P points
- P^2 + P – 1 multipliers (P² for the products between the H_i and X_j, and P – 1 for the products with the exponential)
- P (P – 1) adders
- 1 NCO

As shown in Section 5.2.2, the number of multipliers can be reduced. The optimal reduction provides the minimum number of multipliers, which is 3P – 2 (2P – 1 for the products between the H_i and X_j [16][17], and P – 1 for the products with the exponential). However, for large P, the number of extra adders becomes excessive.

It is then possible to use sub-optimal algorithms that still reduce the number of multipliers while keeping the increase of extra adders moderate [14][15]. Table 1 gives the complexity for the first values of P. It can be seen that for P = 2, the sub-optimal reduction gives the same result as the optimal reduction.

Table 1 : Complexity of different FFAs

	P	Number of complex multipliers	Number of complex adders
No reduction of multipliers	2	5	2
	3	11	6
	4	19	12
Sub-optimal reduction of multipliers	2	4	5
	3	8	13
	4	13	25
Optimal reduction of multipliers	2	4	5
	3	7	25
	4	10	78

5.4 Application to reduce resources

The FFA algorithm can also be used to reduce the resources when time multiplexing is applied. An example is shown in Fig. 13, with the corresponding timing diagram in Fig. 14. The combination algorithm to obtain $Y_0[k]$ and $Y_1[k]$ can be one of the previously presented (Eq. (9) or (11)) or an equivalent one, this is why the value of M is not specified in the caption of Fig. 13.

First the inputs $s_0[n]$ and $s_1[n]$ take the value of the code replica, $h_0[n]$ and $h_1[n]$, their FFT is computed and stored in memory. During the storage, the inputs take the value of the input signal, $x_0[n]$ and $x_1[n]$. When their FFTs are available, the memories are read, the products and combination between the $H_i[k]$ and the $X_i[k]$ are performed to obtain $Y_0[k]$ and $Y_1[k]$. The IFFT of $Y_0[k]$ is computed while $Y_1[k]$ is stored in memory. Then the memory is read and the IFFT of $Y_1[k]$ is computed.

This architecture requires only 3 N/2-point FFTs and two memories ($Y_1[k]$ can be stored in one of the memories used for $H_0^*[k]$ and $H_1^*[k]$ because the writing/reading accesses do not overlap). The throughput of this architecture is identical to the one of the traditional architecture, and the latency is slightly reduced, since the K^th correlation result is available after KN+3N/2+2L.

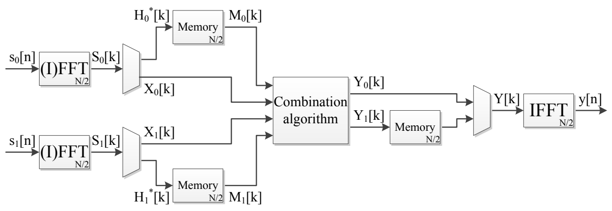

Fig. 13 : FFA-based architecture to reduce resources (architecture 1−3−N/2−M)

[1] This figure contains two typo errors : $Y_0[k]$ and $Y_1[k]$ are inverted, as well as $y_0[n]$ and $y_1[n]$.

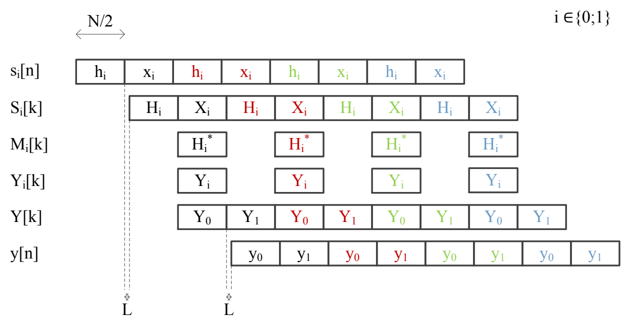

Fig. 14 : Timing diagram of the FFA-based architecture to reduce resources (architecture 1−3−N/2−M)

6. REDUCTION OF THE ENERGY CONSUMPTION

The proposed architectures also have a positive impact on the energy consumption compared to the traditional architecture. The energy consumption is the product of the processing time with the power consumption. The latter is proportional to the resources with a 1 to 1 ratio.

Consequently, the reduction of the power consumption is obtained naturally for the architectures that have reduced resources. Since the processing time is unchanged, the energy consumption is also reduced.

For the architectures that reduce the processing time, if the increase of resources is lower than the decrease of the processing time, the energy consumption will also be reduced. For example, if we consider an architecture that halves the processing time while the resources are increased by 50 %, its energy consumption will be 75 % of that of the initial architecture.

7. APPLICATION EXAMPLE

This section presents a comparison of the different architectures when implemented on FPGAs. A low-cost FPGA family (Altera Cyclone III EP3C120) and a high-end FPGA family (Altera Stratix III EP3SE260) are investigated. The GPS L1 C/A signal is considered. The main lobe of this signal is within 2.046 MHz, we consider thus a sampling frequency of 2.048 MHz, i.e. N = 2048.

We first use models to determine the resource usage of the different functions in the architectures (FFT, multiplier, adder, memory). Except for the FFT, whose model is based on estimates from Altera, the models are quite straightforward and have been checked empirically. Then, a real implementation of two architectures is done to verify the accuracy of the models.

7.1 FPGA Resources

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

- Logic block : This is a small block containing a Look-Up Table (LUT) enabling 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 small size memory (typically between 0.5 and 128 Kibit), having multiple ports.
- Digital Signal Processing (DSP) block : This is a block containing several hardware multipliers (typically 18 × 18 bits).

To compare the different architectures, we will thus consider the three above elements.

In addition to the resource usage of the architectures, we also consider the product resource-processing time, which corresponds to the energy consumption. Thus, to compare fairly the architectures, we define the energy efficiency as the ratio of the energy consumption of the traditional architecture over the energy consumption of the considered architecture :

$$e = \frac{E_{ref}}{E} = \frac{R_{ref} T_{ref}}{RT}, \qquad (12)$$

where E_{ref}, R_{ref}, T_{ref}, E, R, and T are the energy consumption, the resource and the processing time of the traditional architecture and of the considered architecture, respectively. The traditional architecture has thus an energy efficiency of 1. For the other architectures, the greater is the value, the more efficient is the architecture.

7.2 Model for FFT

Altera proposes several ways to implement an FFT [18]. Considering only the streaming implementation and not the buffered ones (because the computation speed is the core of the study), there are two options, fixed streaming, and variable streaming.

Fixed streaming uses the block-floating point arithmetic, receives and outputs the data in natural order only, and can implement a complex multiplication using 4 real multipliers and 2 real adders (conventional representation), or using 3 real multipliers and 5 real adders (canonical representation).

Variable streaming uses a fixed point arithmetic, and has the possibility to receive or output data in bit-reversed order [8][12]. This is a great advantage since it economizes memory resources and reduces the latency. However, variable streaming does not offer the choice of the implementation of complex multiplications.

In Appendix C, we provide a table that summarizes the logic, memory and DSP resources consumption for Stratix III FPGAs for different transform lengths and resolutions. This is an estimation obtained from the Altera Wizard. The real resource consumption will depend on the FPGA chosen, the system implemented and the optimizations selected. However, some observations can be made.

- Doubling the transform length roughly doubles the memory resources, except for some cases where the increase is lower.
- For fixed streaming, doubling the transform length does not change the number of DSP elements, except between 1024 and 2048 points where the number is doubled.
- For the variable streaming, in half of the cases doubling the transform length does not change the number of DSP elements, for the other half there is an increase between 16 % and 50 %.
- Regarding the logic elements, doubling the transform length increase the resources by 11 % on average.

From these observations, it can be inferred that the performance of the different architectures will vary according to the initial transform length and the resolution used.

To be more precise in our estimates, we have measured the resource consumption of the FFT after compilation for the cases we considered, namely 512, 1024 and 2048 points, with a resolution of 18 bits. The table is provided in Appendix C.

7.3 Model for other functions

For the complex multiplications, two models are used. For Stratix III FPGAs, only the conventional representation is possible [19], a complex multiplication requires thus four real multipliers and no extra logic (the two real adders are already included in the DSP blocks). For Cyclone III FPGA, the canonical representation can be used; a complex multiplication requires thus three real multipliers and the equivalent of 8 real adders of R bits, where R is the resolution of the signals used for the complex multiplication.

Regarding the complex addition, it corresponds to two real additions. And a real addition requires R logic elements (LEs, basis block of Cyclone III FPGA), or R/2 adaptive logic module (ALMs, basis block of Stratix III FPGA), where R is the resolution of the signals used for the addition.

Regarding the memory, it is easy to estimate the requirements knowing the number of samples to store and their resolution.

7.4 Results from FPGA models

Based on the previously described models, we have computed the estimated resource usage for different architectures (enumerated in Table 2 to Table 5). For each architecture, we have considered the different possible implementations for the FFT :

- Fixed streaming, and a complex multiplier using 4 real multipliers, denoted as F4.
- Fixed streaming, and a complex multiplier using 3 real multipliers, denoted as F3.
- Variable streaming, and using of natural and bit-reversed order, denoted V.

The F3 case is considered only for Cyclone III FPGAs, since the canonical representation is not available on Stratix III. The case of variable streaming with natural order for the input and the output is not shown because the logic and memory usage is always higher than for the case of variable streaming with natural and bit-reversed order.

The summary of the resource usage and energy efficiency of the architectures is provided for Stratix III FPGAs in Table 2 and Table 3, respectively, and for Cyclone FPGAs in Table 4 and Table 5, respectively. The ALM and LE columns represents the logic resources usage, the M9K column the number of 9216-bit memories used, and the last column the number of DSP 18-bit elements used (1 element corresponding to one 18-bit multiplier).

The traditional architecture is the 1−3−2048−1, with an energy efficiency of 1. From Table 2 and Table 4, it can be seen that the architecture 1−3−1024−4 uses effectively less resources than the traditional one, except for the V implementation of the FFT on Cyclone III FPGAs, which consumes 3% more of LEs. The most efficient architecture regarding the logic usage is the 4−10−512−13; the most efficient regarding the memory usage are the 4−10−512−13 and 4−10−512−10; and the most efficient regarding the DSP usage is the 2−5−1024−4.

It can be noted that the architecture 2−5−1024 uses less memory than the traditional architecture for the F4 and F3 implementations of the FFT. For Stratix III FPGAs, the number of DSP elements is the same, and the logic is increased by only 40 %. For Cyclone III FPGAs, the number of DSP elements is lower considering the F4 implementation of the FFT, and equal considering the F3 implementation of the FFT, for an increase of the logic by 47 % and 35 %, respectively.

Table 2 : Resource usage (model-based) of the architectures for Stratix III FPGA

Architecture (see §3)	Number of ALMs		Number of M9Ks		Number of DSP 18-bit elements	
P−T−N−M	F4	V	F4	V	F4	V
1−3−1024−4	10305	12042	69	51	52	64
1−3−2048−1	11562	12448	117	59	76	64
2−5−2048−2	19332	20749	213	117	128	108
2−6−1024−4	19100	22574	120	84	88	112
2−5−1024−4	16198	19079	109	79	76	96
4−12−512−13	34562	40542	240	132	196	244
4−12−512−10	35516	41496	240	132	184	232
4−10−512−13	29178	34166	209	119	172	212
4−10−512−10	30132	35120	209	119	160	200

Table 3 : Energy efficiency (model-based) of the architectures for Stratix III FPGA. The greater is the better.

Architecture (see §3)	ALMs		M9Ks		DSP 18-bit elements	
P−T−N−M	F4	V	F4	V	F4	V
1−3−1024−4	1.12	1.03	1.70	1.16	1.46	1.00
1−3−2048−1	1	1	1	1	1	1
2−5−2048−2	1.20	1.20	1.10	1.01	1.19	1.19
2−6−1024−4	1.21	1.10	1.95	1.40	1.73	1.14
2−5−1024−4	1.43	1.30	2.15	1.49	2.00	1.33
4−12−512−13	1.34	1.23	1.95	1.79	1.55	1.05
4−12−512−10	1.30	1.20	1.95	1.79	1.65	1.10
4−10−512−13	1.59	1.46	2.24	1.98	1.77	1.21
4−10−512−10	1.53	1.42	2.24	1.98	1.90	1.28

Table 4 : Resource usage (model-based) of the architectures for Cyclone III FPGA.

Architecture	Number of LEs			Number of M9Ks			Number of DSP 18-bit elements		
P−T−N−M	F4	F3	V	F4	F3	V	F4	F3	V
1−3−1024−4	21149	23660	21159	69	69	86	48	39	60
1−3−2048−1	22452	27543	20535	117	117	100	75	57	63
2−5−2048−2	37591	46076	34365	213	213	185	126	96	106
2−6−1024−4	38817	43839	38837	120	120	154	84	66	108
2−5−1024−4	32987	37172	32992	109	109	137	72	57	92
4−12−512−13	72885	82701	70949	240	240	256	183	147	231
4−12−512−10	74361	84177	72425	240	240	256	174	138	222
4−10−512−13	61782	69962	60140	209	209	223	159	129	199
4−10−512−10	63258	71438	61616	209	209	223	150	120	190

Table 5 : Energy efficiency (model-based) of the architectures for Cyclone III FPGA. The greater is the better.

Architecture	LEs			M9Ks			DSP 18-bit elements		
P−T−N−M	F4	F3	V	F4	F3	V	F4	F3	V
1−3−1024−4	1.06	1.16	0.97	1.70	1.70	1.16	1.56	1.46	1.05
1−3−2048−1	1	1	1	1	1	1	1	1	1
2−5−2048−2	1.19	1.20	1.20	1.10	1.10	1.08	1.19	1.19	1.19
2−6−1024−4	1.16	1.26	1.06	1.95	1.95	1.30	1.79	1.73	1.17
2−5−1024−4	1.36	1.48	1.24	2.15	2.15	1.46	2.08	2.00	1.37
4−12−512−13	1.23	1.33	1.16	1.95	1.95	1.56	1.64	1.55	1.09
4−12−512−10	1.21	1.31	1.13	1.95	1.95	1.56	1.72	1.65	1.14
4−10−512−13	1.45	1.57	1.37	2.24	2.24	1.79	1.89	1.77	1.27
4−10−512−10	1.42	1.54	1.33	2.24	2.24	1.79	2.00	1.90	1.33

7.5 Results from real FPGA implementation

To validate our models and conclusion, we have implemented the architectures 1−3−2048−1 and 2−5−1024−4 on a Stratix III FGPA using the F4 implementation of the FFT. Table 6 and Table 7 provide in details the resource consumption of both architectures. The energy efficiency of the architecture 2−5−1024−4 is 1.44 regarding the ALMs, 2.17 regarding the M9K, and 2.00 regarding the DSP elements. As foreseen, this architecture outperforms the traditional one, and the implementation results are very close to the estimated results.

Table 6 : Resource usage of the traditional architecture implemented on Stratix III FPGA

Function	Number of ALMs	Number of M9Ks	Number of DSP 18-bit elements
IFFT (h)	3652	39	24
FFT (x)	3596	39	24
Multiplier	0	0	4
IFFT (y)	3648	39	24
Total	10 896	117	76

Table 7 : Resource usage of the architecture 2−5−1024−4 implemented on Stratix III FPGA

Function	Number of ALMs	Number of M9Ks	Number of DSP 18-bit elements
NCO	1240	0	0
FFT (h)	2748	20	12
Memory	76	8	0
FFT (x_0)	2747	20	12
FFT (x_1)	2741	20	12
Combination	91	0	16
IFFT (y_0)	2779	20	12
IFFT (y_1)	2766	20	12
Total	15 188	108	76

8. CONCLUSIONS

In this paper, we have proposed and compared several alternative architectures to perform the circular correlation using the FFT. Applying these architectures to the Parallel Code-phase Search acquisition of the GPS L1 C/A signal with an FPGA implementation, we have shown that they are more efficient than the traditional one. More specifically : the proposed architecture 1−3−1024−1 offers the same processing time as the traditional architecture (even slightly less due to reduced latency), and reduces at the same time the logic usage by 11 %, the memory by 41 % and the DSP by 32 %, considering the F4 implementation of the FFT on Stratix III FPGA; the architecture 2−5−1024−4 halves the processing time while reducing the memory resources, keeping the same DSP resources, and increasing the logic resources by only 35 %, considering the F3 implementation of the FFT on Cyclone III FPGA; and the architecture 4−10−512−13 divides the processing time by 4 while the memory resources are multiplied by only 1.79, the DSP by 2.26 and the logic by 2.52, considering the F4 implementation of the FFT on Stratix III FPGAs. This architecture also provides the best energy efficiency between the architectures compared, except for the DSP elements, where it is the 2−5−1024−4 that wins. The reason is that the 1024-point FFT uses less multipliers than the 2048-point FFT, whereas the 1024-point FFT and the 512-point FFT use the same number of multipliers (see Table 9). This shows the limit of the method, because for higher decomposition, the efficiency for DSP and logic block will stop increasing due to additional multipliers and adders required for the combination.
Note that the proposed methodology can be applied to any system performing circular correlation.
For future work, we will consider its application to the other GNSS signals, together with others techniques, such as the overlap-and-add method if the sampling frequency does not enable a direct use of a fast algorithm for the DFT.

ACKNOWLEDGMENTS

The authors would like to thank Pradyumna Ayyalasomayajula, Patrick Stadelmann and Youssef

Tawk for their fruitful comments that improved the quality of this paper.

REFERENCES

[1] M. Teixeira, M. De Jesus, and Y. Rodriguez, “Parallel FFT and parallel cyclic convolution algorithms with regular structures and no processor intercommunication,” High Performance Embedded Computing 2005, Lincoln Labs, Lexington, MA, Sept. 2005.

[2] E. Kaplan, C. Hegarty, “Understanding GPS: principles and applications”, Artech House, 2005.

[3] K. Borre, D. Akos, N. Bertelsen, P. Rinder, S.K. Jensen, “A software-defined GPS and Galileo receiver: a single frequency approach”, Birkhäuser Boston, 2007.

[4] F. van Diggelen, “A-GPS: Assisted GPS, GNSS and SBAS”, Artech House, 2009.

[5] 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.

[6] H. Mathis, P. Flammant, A. Thiel, “An analytic way to optimize the detector of a post-Correlation FFT acquisition algorithm”, ION GPS/GNSS 2003, Portland, Oregon, USA, 9-12 Sept. 2003.

[7] 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.

[8] J. Leclère, C. Botteron, P.-A. Farine, “Comparison framework of FPGA-based GNSS signals acquisition architectures”, IEEE Transactions on Aerospace and Electronic Systems, conditionally accepted.

[9] A.V. Oppenheim, R.W. Schafer, “Discrete-time signal processing”, Prentice Hall, 2009.

[10] E.O. Brigham, “The fast Fourier transform and its applications”, Prentice Hall, 1988.

[11] J.O. Smith III, “Mathematics of the discrete Fourier transform (DFT), with audio applications”, W3K Publishing, 2007.

[12] R.G. Lyons, “Understanding digital signal processing ”, Prentice Hall, 2010.

[13] P.P. Vaidyanathan, “Multirate digital filters, filter banks, polyphase networks, and applications: a tutorial”, Proceedings of the IEEE, vol. 78, no. 1, pp. 56-93, Jan 1990.

[14] Z.-J. Mou, P. Duhamel, “Short-length FIR filters and their use in fast nonrecursive filtering”, IEEE Transactions on Signal Processing, vol. 39, no. 6, pp. 1322-1332, June 1991.

[15] D.A. Parker, K.K. Parhi, “Low-area/power parallel FIR digital filter implementations”, The Journal of VLSI Signal Processing, vol. 17, no. 1, pp. 75-92, Sept. 1997.

[16] S. Winograd, “Arithmetic complexity of computation ”, CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM Publications, 1980.

[17] H.J. Nussbaumer, “Fast Fourier transform and convolution algorithms”, Springer Series in Information Sciences, 1982.

[18] Altera, “FFT megacore function user guide”, 2011.

[19] Altera, “Integer arithmetic megafunctions user guide”, 2012.

APPENDIX A : CODE REPLICA GENERATION

A.1 Traditional generation

An NCO is a counter with a step specifying the frequency of the output signal, as shown by Eq. (13), where M is the step, B the number of bits used for the counter, and f_s the sampling frequency, at which runs the NCO [2].

$$f_{code} = \frac{M}{2^B} f_s \quad \Leftrightarrow \quad M = \frac{f_{code}}{f_s} 2^B \tag{13}$$

At each overflow of the counter, a new chip of the PRN code is generated. The implementation of an NCO is shown in Fig. 15, and the timing diagram in Fig. 16 with f_s = 2.048 MHz, f_{code} = 1.023 MHz, B = 32 and thus M = 2,145,386,496.

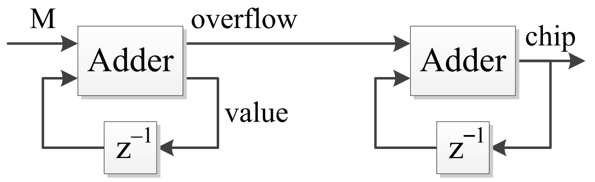

Fig. 15 : Implementation of an NCO

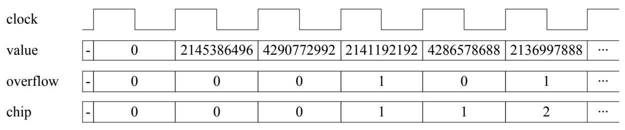
clock
value - 0 2145386496 4290772992 2141192192 4286578688 2136997888 ···
overflow - 0 0 0 1 0 1 ···
chip - 0 0 0 1 1 2 ···

Fig. 16 : Timing diagram of an NCO

A.2 Parallel generation of even and odd samples

From Fig. 17, it can be seen that to generate simultaneously even and odd samples of the replica, we need two NCOs with different starting values and the same increment, 2M. The corresponding schematic is shown in Fig. 18, where the value at the bottom right of the adder is the value of the output at reset.

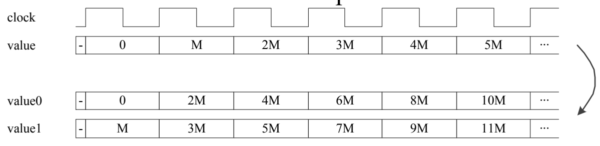
clock
value - 0 M 2M 3M 4M 5M ···
value0 - 0 2M 4M 6M 8M 10M ···
value1 - M 3M 5M 7M 9M 11M ···

Fig. 17 : Timing diagram of an NCO generating even and odd samples simultaneously

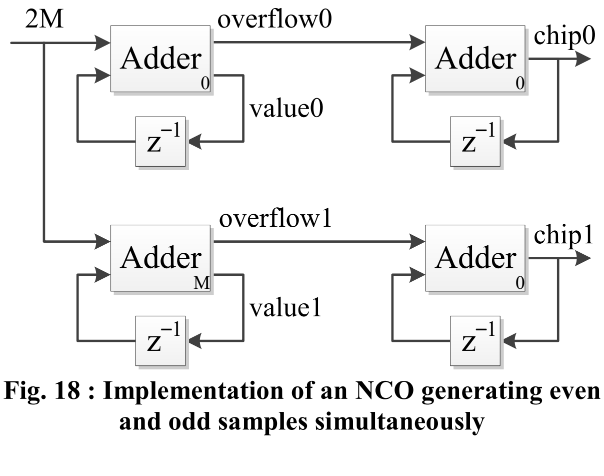

Fig. 18 : Implementation of an NCO generating even and odd samples simultaneously

A.3 Parallel generation of 2 consecutive periods

If the number of samples per code period is an integer, such as in the previous case where there are exactly 2048 samples during one period composed of 1023 chips, the samples of the different periods will be identical, i.e. the j^{th} samples of any period is the same as the j^{th} sample of the first period. Consequently, two consecutive periods are identical and a classical NCO can be used.

If the number of samples per code period is not an integer, the samples of the different periods will be different. Consequently, this requires a modified NCO to generate two consecutive periods.

After 1 cycle, the NCO value is M mod 2^B , after 2 cycles it is 2M mod 2^B, and thus after k cycles it is kM mod 2^B. If $k=2^K$, this means that the value is shifted to the left K times. The modulo operation with 2^B means that we keep the B least significant bits (LSBs) of the value. We can thus infer the NCO value after k cycles, by taking the B-K LSBs of the increment, and shifting it K times (or shifting the increment K times and taking the B LSBs of the result).

It thus requires two NCO based on the same increment, with different starting values. Fig. 19 shows the timing diagram with f_s = 2.048 MHz, f_{code} = 1.023001 MHz, B = 32 and thus M = 2,145,388,593 and M_0 = kM mod 2^B = 4294656. For the next periods, the starting value of the adder should be updated with $2M_0$ and $3M_0$, then with $4M_0$ and $5M_0$, etc., which requires each times two additions. The corresponding schematic is shown in Fig. 20.

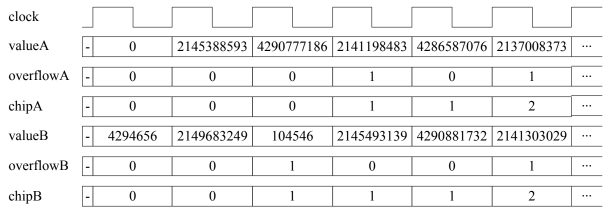
clock
valueA - 0 2145388593 4290777186 2141198483 4286587076 2137008373 ···
overflowA - 0 0 0 1 0 1 ···
chipA - 0 0 0 1 1 2 ···
valueB - 4294656 2149683249 104546 2145493139 4290881732 2141303029 ···
overflowB - 0 0 1 0 0 1 ···
chipB - 0 0 1 1 1 2 ···

Fig. 19 : Timing diagram of an NCO generating consecutive periods simultaneously

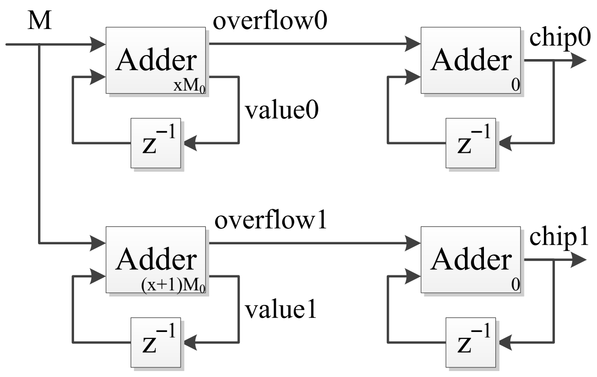

Fig. 20 : Implementation of an NCO generating consecutive periods simultaneously

APPENDIX B

Let us assume that $h_0[n]$ and $h_1[n]$ are two real sequences of N points, and $H_0[k]$ and $H_1[k]$ are their corresponding DFT. We can create a new sequence $h[n] = h_0[n] + j\, h_1[n]$, H[k] being its corresponding DFT. $H_0[k]$ and $H_1[k]$ can be obtained from H[k], as shown by the following equations [12].

$$H_0[k] = \frac{H^*[N-k]+H[k]}{2} = \frac{\mathrm{Re}\left[H[N-k]\right]+\mathrm{Re}\left[H[k]\right]}{2} + j\,\frac{\mathrm{Im}\left[H[k]\right]-\mathrm{Im}\left[H[N-k]\right]}{2} \qquad (14)$$

$$H_1[k] = j\,\frac{H^*[N-k]-H[k]}{2} = \frac{\mathrm{Im}\left[H[k]\right]+\mathrm{Im}\left[H[N-k]\right]}{2} + j\,\frac{\mathrm{Re}\left[H[N-k]\right]-\mathrm{Re}\left[H[k]\right]}{2} \qquad (15)$$

Regarding the hardware implementation, since we have to add and subtract one sequence with its N-k reverse, we need to buffer the data. The corresponding timing diagram is shown in Fig. 21. It can be seen that the writing of the samples of the second period starts while the reading of the reversed samples of the first period is not yet finished. This implies the use of two memories of N complex words, with a write access and a double read access, which will be written and read alternatively. The corresponding schematic is shown in Fig. 22. The combination block is composed of four real adders, equivalent to two complex adders, according to Eqs. (14) and (15). Note that this implementation requires an extra latency of N cycles compared to the use of two FFTs.

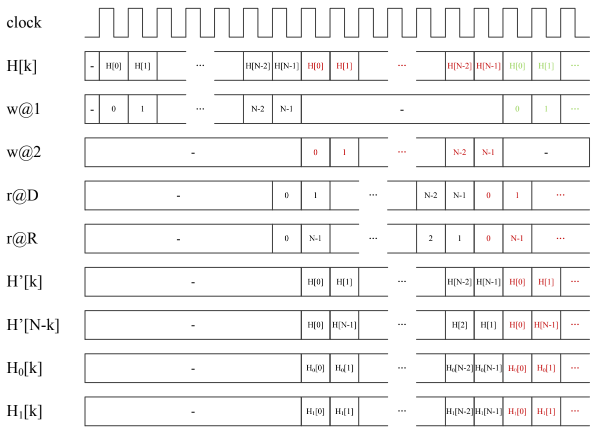
clock
H[k] - H[0] H[1] ··· H[N-2] H[N-1] H[0] H[1] ··· H[N-2] H[N-1] H[0] H[1] ···
w@1 - 0 1 ··· N-2 N-1 - 0 1 ···
w@2 - 0 1 ··· N-2 N-1 -
r@D - 0 1 ··· N-2 N-1 0 1 ···
r@R - 0 N-1 ··· 2 1 0 N-1 ···
H'[k] - H[0] H[1] ··· H[N-2] H[N-1] H[0] H[1] ···
H'[N-k] - H[0] H[N-1] ··· H[2] H[1] H[0] H[N-1] ···
H0[k] - H0[0] H0[1] ··· H0[N-2] H0[N-1] H0[0] H0[1] ···
H1[k] - H1[0] H1[1] ··· H1[N-2] H1[N-1] H1[0] H1[1] ···

Fig. 21 : Timing diagram to perform two real FFTs using one complex FFT

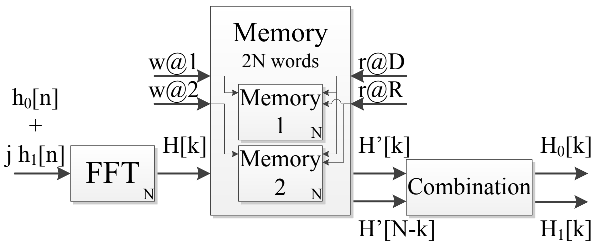

Fig. 22 : Schematic to perform two real FFTs using one complex FFT

APPENDIX C

Table 8 : FFT resources on Stratix III FPGA estimated by the Altera Wizard

Implementation	**Number of points**	**Number of ALUTs (Adaptive LUT, 2 ALUTs = 1 ALM)**				**Number of M9Ks**				**Number of DSP 18-bit elements**			
		12 bit	**14 bit**	**16 bit**	**18 bit**	**12 bit**	**14 bit**	**16 bit**	**18 bit**	**12 bit**	**14 bit**	**16 bit**	**18 bit**
F4	**256**	2736	3048	3591	4265	11	11	11	11	24	24	24	24
	512	2996	3368	3969	4636	11	11	11	11	24	24	24	24
	1024	3435	3864	4523	5248	19	19	19	19	24	24	24	24
	2048	4256	4744	5692	6906	38	38	38	38	48	48	48	48
	4096	4570	5093	6077	7326	57	76	76	76	48	48	48	48
	8192	4400	4888	5836	7050	114	133	152	152	48	48	48	48
	16384	4720	5243	6227	7477	209	247	285	304	48	48	48	48
	32768	4544	5032	5980	7194	418	475	551	608	48	48	48	48
V (natural to bit-reversed)	**256**	4089	4483	4877	5271	1	2	2	2	16	16	20	24
	512	4856	5296	5736	6176	2	3	3	3	24	24	28	32
	1024	5460	8930	6400	6870	4	5	6	6	24	24	28	32
	2048	6211	6735	7259	7783	8	9	11	12	32	32	36	40
	4096	6860	7418	7976	8534	16	18	21	23	32	32	36	40
	8192	7348	7906	8464	9022	32	36	41	45	40	40	44	48
	16384	7848	8406	8964	9522	63	72	81	90	40	40	44	48
	32768	9297	10043	10789	11535	125	143	161	179	48	48	52	56
V (bit-reversed to natural)	**256**	4089	4483	4877	5271	2	2	2	2	16	16	20	24
	512	4856	5296	5736	6176	3	4	4	4	24	24	28	32
	1024	5460	8930	6400	6870	7	7	8	8	24	24	28	32
	2048	6211	6735	7259	7783	13	14	15	17	32	32	36	40
	4096	6860	7418	7976	8534	27	29	31	34	32	32	36	40
	8192	7348	7906	8464	9022	57	61	66	70	40	40	44	48
	16384	7848	8406	8964	9522	117	126	135	144	40	40	44	48
	32768	9297	10043	10789	11535	244	262	274	298	48	48	52	56

Table 9 : FFT resources after compilation using a resolution of 18 bits

Implementation	**Number of points**	**Stratix III FPGA**			**Cyclone III FPGA**		
		Number of ALMs	**Number of M9Ks**	**Number of DSP 18-bit elements**	**Number of LEs**	**Number of M9Ks**	**Number of DSP 18-bit elements**
F4	**512**	2735	20	12	5637	20	12
	1024	3953	20	12	5932	20	12
	2048	3854	39	24	7436	39	24
F3	**512**	-	-	-	6455	20	9
	1024	-	-	-	6769	20	9
	2048	-	-	-	9133	39	18
V (natural to bit-reversed)	**512**	3231	11	16	5490	21	16
	1024	3546	14	16	5947	26	16
	2048	4209	19	20	6828	33	20
V (bit-reversed to natural)	**512**	3238	11	16	5447	22	16
	1024	3504	14	16	5912	25	16
	2048	4030	21	20	6735	34	20