# Architecture of a Real-Time Platform Independent GPS L1 Software Receiver

THÈSE N$^{O}$ 4832 (2010)

PRÉSENTÉE LE 12 NOVEMBRE 2010

À LA FACULTÉ SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE D'ÉLECTRONIQUE ET TRAITEMENT DU SIGNAL
SECTION DE MICROTECHNIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Grégoire WAELCHLI

acceptée sur proposition du jury:

Prof. Ph. Renaud, président du jury
Prof. P.-A. Farine, Dr C. Botteron, directeurs de thèse
Prof. E. Firouzi, rapporteur
Dr D. Manetti, rapporteur
Dr J.-L. Nagel, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Lausanne, EPFL
2010

# Résumé

Les assistants personnels numériques ou les téléphones mobiles modernes ne se limitent plus aux fonctions basiques de communication, mais font également office de récepteurs pour le Système de Positionnement Global (GPS). Le marché croissant d'appareils portables qui intègrent ces fonctionnalités stimule le développement d'une nouvelle génération de récepteurs à base logicielle. Au lieu d'intégrer une puce électronique supplémentaire dédiée au traitement du signal GPS, l'objectif est d'exploiter le microprocesseur déjà présent dans l'appareil afin d'y programmer les fonctionnalités du récepteur. L'intérêt d'une telle solution réside dans l'économie réalisée en termes d'espace et de coûts. Elle offre de surcroît une grande flexibilité et peut facilement évoluer par une simple mise à jour du logiciel, dans la perspective de futurs développements. Comparée à un récepteur matériel, dont la structure est figée, une solution logicielle constitue donc un précieux outil de recherche.

Grâce aux processeurs embarqués toujours plus performants, il devient imaginable de programmer un récepteur logiciel offrant des performances équivalentes à celles d'un récepteur traditionnel. Cependant, au vue de la complexité des différentes opérations à effectuer dans le traitement du signal GPS, l'implantation directe de l'architecture d'un récepteur classique sur un processeur n'est pas réalisable, même pour le plus puissant des ordinateurs. Ainsi, de nouvelles stratégies sont à considérer pour diminuer le nombre d'opérations à réaliser, et par conséquent la charge du processeur. Une première solution consiste à utiliser des micro-instructions spécifiques qui permettent au processeur de traiter plusieurs données à la fois. La parallélisation des opérations diminue leur temps d'exécution de façon significative, mais restreint la portabilité de la solution, optimisée pour un type particulier de processeur. Une alternative exploite la nature même du signal, représenté sous la forme de un ou plusieurs bits d'information.

Une opération complexe, nécessitant un temps de calcul important, peut ainsi être décomposée en plusieurs opérations logiques qui s'effectuent très rapidement entre ces différents bits. L'inconvénient inhérent à cette méthode est son manque de flexibilité puisque son efficacité devient dépendante de la structure du signal et varie donc selon la configuration du récepteur.

Ce travail de thèse s'est inscrit dans le cadre d'un partenariat industriel d'une durée de deux ans (2007-2009) avec l'entreprise *U-blox AG* à Thalwil. Il a eu pour objectif le développement d'un récepteur logiciel GPS L1 fonctionnant en temps réel. Le principal défi de ce projet a consisté à fournir des performances équivalentes à celles d'un récepteur conventionnel, tout en conservant une grande flexibilité afin de permettre la programmation du logiciel sur tout type de processeur. Dans cette perspective, de nouveaux algorithmes ont été développés afin d'optimiser au maximum le nombre, ainsi que la complexité des opérations à effectuer, en vue de diminuer la charge du processeur. Le principe général consiste à regrouper les données dont certaines caractéristiques sont identiques et de les manipuler par lots plutôt qu'individuellement. Ainsi, il devient possible de réduire progressivement le flux d'information à traiter, et par conséquent le nombre d'opérations à effectuer. Une toute nouvelle architecture de récepteur a été proposée et validée à travers la réalisation d'un prototype fonctionnel, démontrant de ce fait la faisabilité du concept.

Mots-clés:

*GPS, récepteur logiciel, temps réel, traitement par lot, portabilité, nouvelle architecture, charge de calcul.*

# Abstract

Personal digital assistants or mobile phones applications are not anymore restricted to multimedia or wireless communications, but have been extended to handle Global Positioning System (GPS) functionalities. Consequently, the growing market of GPS capable mobile devices is driving the interest of software receiver solutions as they provide several advantages with respect to traditional hardware implementations. First, they share the same system resources such as the processor, embedded memory and power with other system units, reducing both the size and the costs of their integration. Second, they can be easily reprogrammed - via a firmware update - for incorporating the latest developments, such as the exploitation of the future satellites signals or some improved multipath mitigation techniques. Finally, they offer a more flexible solution for rapid research and development as compared to conventional hardware receivers where the chip design is fixed and obtained after a long integration process.

With the increasing performance of modern processors, it becomes now feasible to implement in software a multi-channel GPS receiver operating in real-time. However, a major problem with the software architecture is the large computing resources required for the digital signal processing. Former studies have demonstrated that a straightforward transposition of traditional hardware based architectures into software would lead to an amount of integer operations which is not suitable for today's fastest computers. From this observation, several strategies have been proposed in the literature in order to reduce the complexity of the receiver operations. The first one relies on the utilization of advanced microprocessor instructions set which provides the capability of processing vectors of data by operating on multiple integer values at the same time. This results in significant gains in execution speed, but also severely limits the portability of the code, since the operations are tied to specific processors architectures.

Another alternative consists in exploiting the native bitwise representation of the signal. The data bits are stored in separate vectors on which logical parallel operations can be performed. The objective is to take advantage of the universality, high parallelism, and speed of the bitwise operations for which a single integer operation translates into a few simple parallel logical relations. However, the inherent drawback of the bitwise processing is the lack of flexibility as the complexity becomes bit-depth dependent.

This thesis has been carried out in the framework of a two-year industrial project (2007-2009) in collaboration with *U-blox AG* in Thalwil. It aimed to the realization of a multi-channel, platform-independent real-time GPS L1 software receiver. The main challenge of this project consisted in providing real-time performances while keeping the portability of the code to make the receiver suitable for any type of software implementation. In that sense, new techniques and algorithms have been developed for optimizing the processing chain in order to lower the processor load. The main idea consists in regrouping data which share the same characteristics, and process them in batches instead of sequentially. This way, it becomes possible to progressively reduce the data throughput and consequently the amount of operations to perform. A completely new receiver architecture has been proposed and validated through the realization of a functional prototype, thus demonstrating the feasibility of the concept.

Keywords:

*GPS, software receiver, real-time, batch processing, platform-independent, new architecture, computational load*

# Acronyms

| | | |
|---|---|---|
| **ADC** | Analog to Digital Converter | 2 |
| **ASIC** | Application Specific Integrated Circuit | 2 |
| **BOC** | Binary Offset Carrier (m,n) | 20 |
| **BPSK** | Binary Phase Shift Keying | 18 |
| **CA** | Coarse Acquisition | 11 |
| **CBOC** | Composite BOC | 21 |
| **CDMA** | Code Division Multiple Access | 16 |
| **CPU** | Central Processing Unit | 41 |
| **CS** | Commercial Service | 20 |
| **DC** | Direct Current | 124 |
| **DFT** | Discrete Fourier Transform | 38 |
| **DLL** | Delay Lock Loop | 48 |
| **DSP** | Digital Signal Processor | 66 |
| **E** | Early | 31 |
| **FFT** | Fast Fourier Transform | 37 |
| **FLL** | Frequency Lock Loop | 48 |
| **FPGA** | Field Programmable Gate Array | 4 |
| **GLONASS** | Global Orbiting Navigation Satellite System | 3 |
| **GNSS** | Global Navigation Satellite System | 1 |
| **GPS** | Global Positioning System | 1 |
| **HOW** | Handover Word | 15 |
| **IF** | Intermediate Frequency | 23 |
| **L** | Late | 31 |

| | |
|---|---|
| **L1** | Link 1 ..... 11 |
| **L2** | Link 2 ..... 11 |
| **LNA** | Low-Noise Amplifier ..... 23 |
| **LO** | Local Oscillator ..... 58 |
| **LUT** | Look Up Table ..... 70 |
| **NCO** | Numerically Controlled Oscillator ..... 28 |
| **OS** | Open Service ..... 20 |
| **P** | Prompt ..... 31 |
| **P(Y)** | Precise ..... 11 |
| **PC** | Personal Computer ..... 123 |
| **PCB** | Printed Circuit Board ..... 123 |
| **PDA** | Personal Digital Assistant ..... 1 |
| **PLL** | Phase Lock Loop ..... 48 |
| **PRN** | Pseudo Random Noise ..... 15 |
| **PRS** | Public Regulated Service ..... 20 |
| **PVT** | Position - Velocity - Time ..... 2 |
| **RAM** | Random Access Memory ..... 153 |
| **RF** | Radio Frequency ..... 2 |
| **SIMD** | Single Instruction Multiple Data ..... 63 |
| **SNR** | Signal-to-Noise Ratio ..... 20 |
| **SoL** | Safety of Life Service ..... 20 |
| **SPI** | Serial Peripheral Interface ..... 56 |
| **SSE** | Streaming SIMD Extension ..... 64 |
| **TCXO** | Temperature Controlled Crystal Oscillator ..... 59 |
| **TLM** | Telemetry word ..... 15 |
| **TOW** | Time Of Week ..... 15 |
| **USB** | Universal Serial Bus ..... 56 |
| **VGA** | Variable Gain Amplifier ..... 23 |
| **VHDL** | Very high speed integrated circuit High Density Language 124 |
| **XO** | Crystal Oscillator ..... 59 |

# Nomenclature

| Operators and functions | |
|---|---|
| & | Logical AND |
| \| | Logical OR |
| $\wedge$ | Logical XOR |
| $\sim$ | Logical complement |
| $\gg$ | Logical right shift |
| $\ll$ | Logical left shift |
| $\mathrm{Re}(x)$ | Real part of $x$ |
| $\mathrm{Im}(x)$ | Imaginary part of $x$ |
| $\lfloor x \rfloor$ | floor($x$) function that rounds the value of $x$ towards the nearest smallest integer |
| $\lceil x \rceil$ | ceil($x$) function that rounds the value of $x$ towards the nearest largest integer |
| $O(x)$ | Operator counting the number of bits equal to '1' in the word $x$ |

| Constants | |
|---|---|
| $\pi \approx 3.14$ | Pi |
| $c \approx 3 \cdot 10^8$ | Speed of light [m/s] |
| $k \approx 1.38 \cdot 10^{-23}$ | Boltzmann constant [J/K] |

Variables

| | |
|---|---|
| $\omega_{L1}$ | GPS L1 angular frequency [Hz] |
| $\omega$ | Carrier angular frequency (plus Doppler) [Hz] |
| $\widetilde{\omega}$ | Estimated carrier angular frequency (plus Doppler) [Hz] |
| $\Delta\omega$ | Carrier angular frequency mismatch [Hz] |
| $\omega_{0d}$ | Natural radian frequency of the DLL [Hz] |
| $\omega_{0f}$ | Natural radian frequency of the FLL [Hz] |
| $\omega_{0p}$ | Natural radian frequency of the PLL [Hz] |
| $\phi_{L1}$ | GPS L1 carrier phase [rad] |
| $\phi$ | Carrier phase [rad] |
| $\widetilde{\phi}$ | Estimated carrier phase [rad] |
| $\Delta\phi$ | Carrier phase mismatch [rad] |
| $\rho$ | Pseudorange [m] |
| $\tau$ | Offset of the local code replica [chips] |
| $\overline{a}$ | Average number of samples per carrier interval |
| $a_j$ | Carrier samples accumulator |
| $a_{off}$ | Carrier samples accumulator offset |
| $A_j$ | Samples boundaries of the carrier interval $j$ |
| $\overline{b}$ | Average number of samples $\overline{b}$ per chip |
| $b_k$ | Code samples accumulator |
| $b_{off}$ | Code samples accumulator offset |
| $\overline{B}$ | Constant chip sums summation interval |
| $B_k^x$ | Samples boundaries of the chip $k$ ($x \in E,\ P,\ L$) |
| $B_c$ | GPS L1 CA code null-to-null bandwidth [Hz] |
| $B_n$ | Equivalent noise bandwidth of the signal [Hz] |

| | |
|---|---|
| $B_{nd}$ | Noise bandwidth of the DLL [Hz] |
| $B_{nf}$ | Noise bandwidth of the FLL [Hz] |
| $B_{np}$ | Noise bandwidth of the PLL [Hz] |
| $c(t)$ | Continuous GPS L1 CA code |
| $c(n)$ | Discrete GPS L1 CA code |
| $c^x(n)$ | E, P, and L local code replicas ($x \in E,\ P,\ L$) |
| $C(k)$ | Local code replica ($k^{th} chip$) |
| $d$ | Distance between the satellite and the receiver [m] |
| $d(t)$ | Continuous GPS L1 data message |
| $d(n)$ | Discrete GPS L1 data message |
| $D_f$ | Carrier freqeuency discriminator [Hz] |
| $D_\phi$ | Carrier phase discriminator [rad] |
| $D_\tau$ | Code discriminator [chips] |
| $f_{L1}$ | GPS L1 carrier frequency [Hz] |
| $f_{L2}$ | GPS L2 carrier frequency [Hz] |
| $f_c$ | Local code replica frequency [Hz] |
| $\Delta f_c$ | Local code replica frequency drift [Hz] |
| $f_d$ | Residual Doppler frequency [Hz] |
| $f_s$ | Sampling frequency [Hz] |
| $\Delta f_s$ | Sampling frequency drift [Hz] |
| $\Delta f$ | Frequency mismatch [Hz] |
| $\delta f$ | Frequency resolution [Hz] |
| $G$ | Gain |
| $I(t),\ Q(t)$ | Continuous complex signal |
| $I(n),\ Q(n)$ | Discrete complex signal |

| | |
|---|---|
| $I_m(n),\ Q_m(n)$ | $m^{th}$ bit of $I(n)$ and $Q(n)$ |
| $I^x(n),\ Q^x(n)$ | Complex signal after code removal ($x \in E,\ P,\ L$) |
| $I^x_m(n),\ Q^x_m(n)$ | $m^{th}$ bit of $I^x(n)$ and $Q^x(n)$ ($x \in E,\ P,\ L$) |
| $I_{bb}(n),\ Q_{bb}(n)$ | Complex signal after carrier removal |
| $I_{bb,m}(n),\ Q_{bb,m}(n)$ | $m^{th}$ bit of $I_{bb}(n)$ and $Q_{bb}(n)$ |
| $I^x,\ Q^x$ | Complex correlation results ($x \in E,\ P,\ L$) |
| $\widetilde{I^x},\ \widetilde{Q^x}$ | Estimated complex correlation results ($x \in E,\ P,\ L$) |
| $(I^x)',\ (Q^x)'$ | Approximated complex correlation results ($x \in E,\ P,\ L$) |
| $Inc$ | NCO increment |
| $j$ | Index of the carrier interval |
| $J$ | Number of carrier intervals per integration period |
| $k$ | Index of the chip |
| $K$ | Number of chips per integration period |
| $K(n)$ | Complex local carrier replica |
| $K_m(n)$ | $m^{th}$ bit of $K(n)$ |
| $M$ | Number of quantization bits |
| $n$ | Positive integer denoting the $n^{th}$ sample |
| $N$ | Number of points of the FFT |
| $N_b$ | Number of bits per CPU word |
| $N_c$ | Number of satellite channels |
| $N_p$ | Number of pre-detection sums |
| $N_s$ | Number of samples accumulated over the period $T_{int}$ |
| $N_{clk}$ | Number of NCO clock cycles to achieve one carrier period |
| $P^x_j$ | Complex partial sum $j$ ($x \in E,\ P,\ L$) |
| $P_e$ | Probability of having an erroneous sample in a chip sum |

| | |
|---|---|
| $P_n$ | Thermal noise power [dBW] |
| $\overline{r}$ | Average number of chips per carrier interval |
| $r_j$ | Carrier chips accumulator |
| $r_{off}$ | Carrier chips accumulator offset |
| $R_j^x$ | Chips boundaries of the carrier interval $j$ ($x \in E,\ P,\ L$) |
| $R(\Delta\tau)$ | Autocorrelation function of the GPS L1 CA code |
| $sin(j),\ cos(j)$ | Complex carrier magnitude of the carrier interval $j$ |
| $S_n$ | Complex chip sum $n$ |
| $S_m^x$ | Sum of the $m^{th}$ bits of $I^x(n)$ over the period $T_{int}$ |
| $t_{sat}$ | Biased satellite clock reading the time at which the signal leaves the satellite [s] |
| $t_{rec}$ | Biased receiver clock reading the time at which the signal reaches the receiver [s] |
| $\delta t_{sat}$ | Bias error of the satellite clock [s] |
| $\delta t_{rec}$ | Bias error of the receiver clock [s] |
| $T_{sat}$ | Reference time at which the signal leaves the satellite [s] |
| $T_{rec}$ | Reference time at which the signal reaches the receiver [s] |
| $T_d$ | Update time of the DLL [s] |
| $T_p$ | Pre-detection time [s] |
| $T_s$ | Sampling period [s] |
| $T_{code}$ | Code period [s] |
| $T_{int}$ | Integration period [s] |
| $T_{pf}$ | Update time of the PLL-assisted-FLL [s] |
| $\overline{v}$ | Number of chip sums per pre-detection sum |
| $v_i$ | Pre-detection chip sums accumulator |
| $V_i$ | Chip sums boundaries of the pre-detection sum $i$ |
| $W$ | NCO accumulator bit width |

# Contents

# Chapter 1

# Introduction

Personal Digital Assistants (PDAs) or mobile phones applications are not anymore restricted to multimedia or wireless communications, but have been extended to handle Global Navigation Satellite System (GNSS) functionalities. The new generation of portable devices is becoming suitable for use as navigators thanks to their large displays, their improved storage capacities and the increasing number of developers creating software applications for them. Accordingly to an electronics market research [iSu08], the cellphones are expected to replace the personal navigation devices as primary Global Positioning System (GPS) receivers by 2011. One major drive in this fast development can be credited to the US Enhanced 911 mandate, under which the location of any cellphone must be available to emergency call dispatchers since the end of 2005 [Pub10].

But the use of the GNSS technology in the cellular handset has some drawbacks. The functionalities are usually implemented using a standalone hardware module, introducing a relatively expensive item to integrate into such cost sensitive devices. However, with the increasing performance of modern embedded processors, it becomes now feasible to implement a GNSS receiver in software, where all the basic operations are performed on a general purpose microprocessor. In that case, both the receiver and the device can share the same system resources, reducing both the size and the costs of their integration. Consequently, the fast growing market of GNSS capable mobile devices is driving the interest of the software receiver solutions, as they present many advantages, especially in term of costs, with respect to the traditional hardware implementations.

## 1.1 Principle of a software receiver

In a generic GNSS receiver, the satellites signals traveling through space are first received by the antenna, then properly conditioned through the Radio Frequency (RF) front-end, before being digitized by means of an Analog to Digital Converter (ADC). The resulting data samples are then transmitted to the receiver digital processing unit in order to extract all the information necessary to compute the Position - Velocity - Time (PVT) solution. The block diagram of a typical GNSS receiver is illustrated in Figure 1.1.

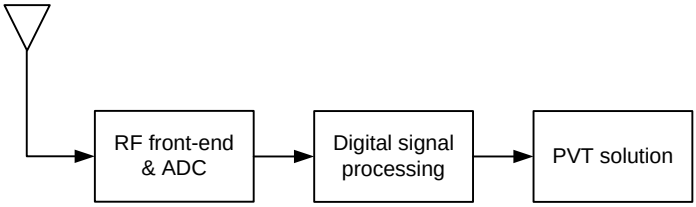


**Figure 1.1:** Generic GNSS receiver block diagram.

In today's most mass market receivers, the complete digital signal processing chain is implemented on an Application Specific Integrated Circuit (ASIC). The use of a hardware chip, designed for this particular task only, is motivated by the power demanding nature of the operations that need to be performed continuously and at high speed.

The software receiver aims to substitute the dedicated hardware chip for a general purpose programmable microprocessor in order to fully process the data samples in software. The primary objective of this approach is to separate the analog signal conditioning (in the hardware RF front-end) from the digital signal processing (in software), so that any low-level receiver functionality modification can be operated by a simple firmware update. The second objective is to sample the signal as close as possible as to the antenna, in order to perform the maximum number of operations on the microprocessor and thus, reduce the hardware part to the minimum, with direct implication on the overall cost.

## 1.2 Benefits of a software receiver

The first gain of a software receiver lies in its powerful flexibility. While the ASIC functionalities are by definition restricted to the particular design of the chip, signal processing in software can easily be reprogrammed via a firmware update for integrating the latest developments. Flexibility and updating capabilities will become even more important in the near future as the world of global navigation is frenetically evolving. For example, the Russian Federation is currently modernizing its Global Orbiting Navigation Satellite System (GLONASS) while the European Union and China are about to introduce their own full satellites constellations (respectively named Galileo and Compass) within the next few years. The users of software receivers will derive full benefit from these new signals with a simple firmware upgrade, without purchasing new hardware components. A software receiver thus constitutes a unique tool for research and development purposes, as it can accommodate different sorts of RF front-ends with various sampling frequencies and data types.

Another key feature of the software approach is the cost saving opportunity. Many handsets now embed a powerful applications host processor for decoding streaming music and video files. The software flexibility allows the device to dynamically change its functionalities and when these services are not in use, the application processor is available to perform GNSS signal processing. This also enables more flexible power management than hardware based approaches, since different parts of the system can be individually controlled. Consequently, in a host system equipped with a microprocessor, both the receiver and the device can share the same resources, such as the memory and the power, reducing both the size and the costs of their integration, since the ASIC is no more needed.

## 1.3 Compromises of a software receiver

Implementing a GNSS receiver in software is not straightforward. The large computing resources required for performing the different operations on a microprocessor constitute a major issue, as compared to a hardware design where the chip is specifically designed for this task and can thus handle much higher data throughput. Several studies (e.g. [Hec06]) demonstrated that a direct transposition of traditional hardware based architectures into software leads to an amount of arithmetic operations which is simply not suitable for real-time applications.

This is the inherent trade-off of the software versus hardware approach, since the flexibility is obtained at the cost of lower processing power capabilities. However, a compromise between the hardware and the full software implementation can be obtained with a design based on Field Programmable Gate Array (FPGA). The latter consists in a re-configurable integrated circuit containing programmable blocks that can be configured to perform any complex logical functions that an ASIC could do. FPGAs are increasingly preferred to microprocessors in conventional high performance applications since their architecture offers massive parallelism logic resources, providing a considerable computational throughput even at a low clock rates. The trade-off between flexibility and processing power of the different receiver designs is illustrated in Figure 1.2.

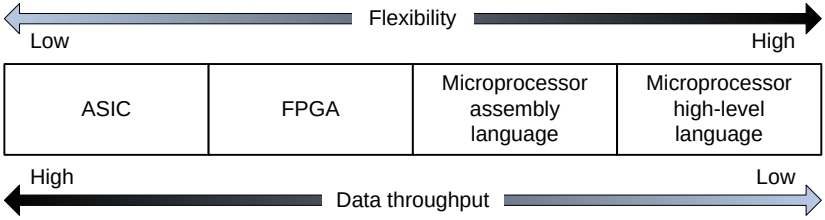


**Figure 1.2:** Trade-off between the high data throughput of the ASIC and the flexibility of the software.

The exact definition of a software receiver always brings some confusion among engineers and many consider the FPGA approach as a full software solution since it is reconfigurable [Bro07], [Pur05] or [Gan04]. However, the terminology "software" usually implies that the processing of the signal is carried out by a programmable microprocessor, excluding the FPGA design, which is generally specified using a hardware description language. Consequently, the FPGA based receivers are not further considered in this work.

## 1.4 History of software receivers

The idea of a receiver performing signal processing in software is not new, but the increasing performances of digital electronics are making practical today many concepts that were only theoretically possible two decades ago. Although the terminology of *Software Defined Radio* was first proposed by Joseph Mitola in 1991 and officially published in 1992 [Mit92], the software receiver finds its origins in projects of defense in the United States and Europe since the late 1970's. One of the first large-scale software receiver initiatives was an US military project named *SpeakEasy*, which was motivated by the communication interoperability problems between the different military branches having dissimilar radio systems [Lac95]. The proliferation of incompatible equipments became a significant logistic burden and the project sought to implement the basic radio functions in a full digital, software programmable base-band signal processor. The objective was to emulate more than 15 different military radios, with the ability of integrating new modulation standards in the future. The radio receiver basic architecture was built around a RF front-end feeding an ADC, the latter being connected to a bank of processors (40 MHz *Texas Instrument*) and FPGAs. The project was successfully demonstrated in 1994 and was the first known to use FPGAs for the digital processing of radio data. However, it involved several hundred of processors that filled the back of a truck and took three years of software development for a platform that became obsolete after a few months only [Bos04]. Furthermore, the software was tied to the specific hardware assembly language, plus all of the specialized glue code to get the processors working together. These observations made that the software portability should become a key goal in the future projects.

One of the first initiatives related to GNSS civilian applications goes back to 1988, with the proposition of a new approach using a general purpose microprocessor to perform the digital processing [Mat88]. The primary objective was to speed-up the time requested to acquire four satellites in a hand-held navigation system receiver. The basic idea was to first process a single satellite at once in order to download the ephemeris data. Then, the receiver would be activated every two or five minutes - to make a quick re-acquisition of the four satellites to find the position - and shutdown, leaving the position displayed in-between two operations. In normal operation, knowing the satellites to search for and, by predicting the clock drift over a few minutes, the worst case acquisition time was less than 12 seconds.

In 1997 the Ohio University developed a software receiver based on a novel bandpass sampling front-end design for processing multiple GNSS transmissions (GPS and GLONASS). The signal was post-processed in software using the *Matlab* programming language and one second of data required about 45 seconds of processing time (200 MHz *Intel Pentium Pro*) [Ako97]. Although a programming in C language would definitively offer improved performances as compared to the *Matlab* implementation, the author raised the question of the ability of a software receiver to ever process the incoming data in real-time. Even though he recognized that this was not feasible at the time, he was quite confident that it would be in the near future, motivated by Moore's law, which states that the processing power doubles every 18 months [Sch96].

Based on the previous work of [Ako97], the Stanford University succeeded in 2001 to implement a complete four channels GPS receiver operating in real-time on a *x86* based processor (650 MHz *AMD Duron*) [Ako01a]. Since then, many universities and companies have been active in the field of software receivers. For example, in 2003, the Cornell University presented a 12 channels GPS L1 software receiver running in real-time on a personal computer (1.7 GHz *AMD Athlon*) [Led03].

More recently, the research activities have focused on the development of software receivers capable of processing the next generation of satellites signals such as Galileo E1-B [Led06a] or GPS L5 [Mon07].

## 1.5 Presentation of the thesis

This Ph.D. thesis work considers the realization of a multi-channel GPS L1 software receiver operating in real-time on a general purpose microprocessor. It focuses mainly on the development of a new base-band architecture for minimizing the computational load of the different operations involved in the digital signal processing chain. The two main challenges of this work consist in:

- operating the receiver in real-time with hardware-like performances and minimal processor load;
- keeping the software portable to make the receiver suitable for any type of processor (platform-independent).

### 1.5.1 Context of the thesis and partnership

This thesis has been carried out in the framework of a two-year industrial project (2007-2009) in collaboration with *u-blox AG*, aiming to the realization of a multi-channel, platform-independent real-time GPS L1 software receiver. The receiver was successfully implemented on a host computer communicating with a front-end unit for receiving the digitized satellites signals, and an external unit integrating the PVT solution computation.

Some contributions to the research activities were also carried out in the framework of several other research and development projects:

- SARBACAN (Safety And Rescue BeAcon development with CANada) for the development of a software architecture able to process the new Galileo E1 modulation, as well as to decode the search and rescue messages.
  Main industrial partners: *KANNAD*, *u-blox AG*, *Novatel*;
- GRDB (Galileo Receiver for Distress Beacon) for the realization of a Galileo receiver prototype able to process the new BOC(1,1) modulation, as well as to decode the search and rescue messages.
  Industrial partner: *Martec Serpe* IESM (MSI) now *KANNAD*.

### 1.5.2 Organization of the report

The objective of the present thesis is to provide to the reader all the information necessary to fully apprehend the challenges of implementing a receiver in software. The document is subdivided into eight chapters, as follows:

- Chapter 2 introduces the basic structure of the broadcasted GPS L1 signal and details the different receiver operations involved in its processing, from the signal reception at the antenna up to the PVT solution computation;
- Chapter 3 identifies the inherent constraints of realizing a receiver in software as compared to a traditional hardware based implementation;
- Chapter 4 reviews the state-of-the-art in the field of software receivers. Different algorithms and architectures proposed in the literature for optimizing the receiver complexity and achieving real-time performances are presented. Their advantages and limitations are also discussed;
- Chapter 5 describes the main contribution of this work and presents new algorithms for minimizing the computational load of the basic receiver operations. Finally it contains the proposition of a completely new receiver architecture suitable for a real-time implementation on a general purpose microprocessor;
- Chapter 6 analyzes the theoretical performance of the proposed architecture in terms of the amount of integer operations to perform each second. The trade-offs and limitations of the receiver are also discussed, as well as its ability to accommodate the next generation of GNSS signals;
- Chapter 7 details the implementation of the proposed architecture in a demonstrator and evaluates the performance of the receiver in terms of accuracy and processor and memory requirements;
- Chapter 8 summarizes the achievements of this thesis.

## 1.6 Contributions of the thesis

Most of the existing software receivers are tied to specific implementations that severely limit their portability or their flexibility. In order to implement a software receiver operating in real-time on different platforms and accommodating various signals configurations, there is a need for a new architecture combining both flexibility and efficiency. In this context, the main contributions of this thesis in the field of the software receivers are:

- development of a completely novel receiver base-band architecture suitable for a real-time software implementation on a general purpose microprocessor;
- development of an algorithm based on batch processing for the real-time generation of the local carrier replicas;
- development of an algorithm based on batch processing for the real-time generation of the local code replicas;
- development of an algorithm based on distributed arithmetic for optimizing the operations involved in the base-band processing;
- realization of a GPS L1 receiver demonstrator operating in real-time on different host computers equipped with various processors.

Part of this thesis work has been the subject of various publications:

Patents:

- C. Buergi, G. Waelchli, and M. Baracchi. "A method of processing a digital signal derived from a direct-sequence spread spectrum signal and a receiver for carrying out the method". European Patent 09405207.3, November 2009;
- C. Buergi, G. Waelchli, and M. Baracchi. "A method of processing a digital signal derived from a direct-sequence spread spectrum signal and a receiver". US Patent 12/694,145, January 2010.

Journals:

- M. Baracchi, G. Waelchli, C. Botteron, and P.-A. Farine. "Real-time GNSS software receiver: challenges, status and perspectives". *My coordinates*, VI(5):7–9, 2010;

- G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Distributed arithmetic for efficient base-band processing in real-time GNSS software receivers". *Hindawi, Journal of Electrical and Computer Engineering*, January, 2010;
- M. Baracchi, G. Waelchli, C. Botteron, and P.-A. Farine. "Real-time GNSS software receiver: challenges, status and perspectives". *GPS World*, 20(9):40–47, 2009.

Conferences:

- G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Batch processing for efficient base-band operations in real-time GNSS software receivers". *Institute of Navigation GNSS 2010*. Portland, OR, USA, September 21-24 2010;
- G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Real-time carrier generation for a GNSS software receiver". *International Symposium on GPS/GNSS 2009*. Jeju, South Korea, November 4-9 2009;
- M. Baracchi, G. Waelchli, C. Botteron, and P.-A. Farine. "Real-time GNSS software receiver: challenges, status and perspectives". *European Navigation Conference on GNSS 09*. Naples, Italy, 3-6 May 2009;
- G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Performances of a new correlation algorithm for a platform-independent GPS software receiver". *2009 International Technical Meeting of the Institute of Navigation*, pp. 1062–1067. Anaheim, CA, USA, January 26-28 2009;
- G. Waelchli, G. Zamuner, D. Manetti, M. Frei, F. Chastellain, E. Firouzi, C. Botteron, P.-A. Farine, and P. Brault. "Development, implementation and validation of a real-time Galileo E1 signal acquisition and tracking scheme". *European Navigation Conference on GNSS 07*, pp. 626–634. Geneva, Switzerland, May 29-June 1 2007.

## 1.7 Summary

Chapter 1 introduces the concept of a software receiver and explains the motivations of implementing the complete digital processing chain on a microprocessor. From this, the objectives of this thesis work are described.

# Chapter 2

# GPS signals & receiver operations

In order to fully apprehend the challenges of a software receiver, it is first necessary to understand the structure of the broadcasted GPS signals, as well as the different receiver operations involved in its processing. A brief overview is given in this chapter.

## 2.1 GPS signals

The GPS is a US navigation system providing positioning and timing services to worldwide users. It relies on a constellation of 24 operational satellites orbiting in six quasi circular planes, approximately 20'200 km above the earth surface. Each satellite broadcasts two signals at the same time, namely Link 1 (L1) and Link 2 (L2), made of three components:

1. a carrier frequency $f_{L1}$ (1'575.42 MHz) or $f_{L2}$ (1'227.6 MHz);
2. a data message containing the information relative to the satellites orbits and necessary to compute the user PVT solution;
3. two unique identification codes, namely Coarse Acquisition (CA) and Precise (P(Y)), proper to each satellite. The CA code, freely accessible, is sent on the L1 carrier only. The P(Y) code, encrypted and restricted to authorized users, is transmitted on both L1 and L2 bands at the same time.

### 2.1.1 GPS signals components

#### GPS carrier frequency

The transmission of data from the satellites to the receiver requires a proper carrier signal with a frequency not influenced by weather phenomena like rain, snow or clouds, and robust to ionospheric delays (more important for frequency ranges below 100 MHz and above 10 GHz). The carrier must also be large enough to allow high bandwidth data modulation with the different identification codes. Based notably on these constraints, the choice of $f_{L1} = 1'575.42$ MHz and $f_{L2} = 1'227.6$ MHz center frequencies is advantageous.

The carrier frequency is affected by the Doppler effect, induced by the relative motion between the satellite and the receiver (see Figure 2.1). The received frequency increases as the satellite approaches and decreases as it recedes from the user. The variation $\Delta f$ of the initial frequency $f_0$ can be estimated as:

$$\Delta f = \frac{v_{s,r}}{c} \cdot f_0 \text{ [Hz]} \tag{2.1}$$

where $v_{s,r} = v_s - v_r$ is the velocity of the satellite $v_s$ relative to the velocity of the receiver $v_r$;
$c \approx 3 \cdot 10^8$ m/s is the speed of light.

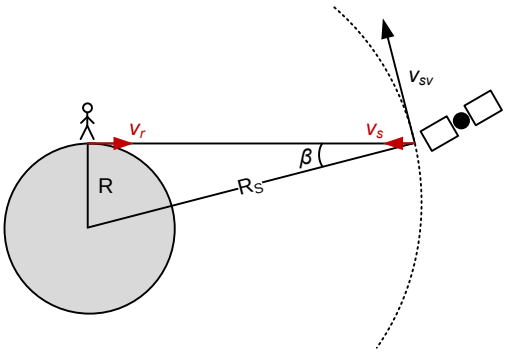


**Figure 2.1:** Doppler effect on the carrier frequency induced by the relative velocity between satellite vehicle $v_s$ and the receiver $v_r$.

The speed $v_{sv}$ of the satellite vehicle can be estimated as:

$$v_{sv} = \frac{2 \cdot \pi \cdot R_s}{T_s} = \frac{2 \cdot \pi \cdot 26'560'000}{11 \cdot 3600 + 58 \cdot 60 + 2} \approx 3874 \text{ m/s} \tag{2.2}$$

where $R_s$=26'560 km is the average radius of the satellite orbit;
$T_s$=11 h, 58 m, 2 s is the orbital time of the satellite.

The Doppler is induced by the velocity component $v_s$ of the satellite vehicle toward the user that can be expressed as:

$$v_s = v_{sv} \cdot \sin(\beta) \approx 3874 \cdot \sin(\beta) \text{ m/s} \tag{2.3}$$

The maximum absolute Doppler is obtained when the satellite elevation is at the horizontal position referenced to the user, corresponding to an elevation angle $\beta_{lim}$ of:

$$\sin(\beta_{lim}) = \frac{R}{R_s} = \frac{6'360}{26'560} \approx \frac{1}{4} \tag{2.4}$$

Consequently, the maximum absolute Doppler velocity, which is along the horizontal direction, becomes:

$$v_{s\ max} = v_{sv} \cdot \sin(\beta_{lim}) \approx 920 \text{ m/s} \tag{2.5}$$

This speed is equivalent to a high-speed military aircraft. In comparison, the additional Doppler shift caused by a terrestrial vehicle is often very small ($v_{r\ max} < 50$ m/s), even if the motion is directly toward the satellite to produce the higher effect [Tsu05]. Consequently, for the GPS L1 frequency and for terrestrial applications, the maximal absolute Doppler shift can reasonably be estimated as:

$$\Delta f_{max} = \frac{v_{s,r\ max}}{c} \cdot f_{L1} = \frac{920 + 50}{3 \cdot 10^8} \cdot 1.5 \cdot 10^9 \approx 5 \text{ kHz} \tag{2.6}$$

The Doppler also affects the code frequency, but in a lesser extent because of the much lower frequency of the CA code as compared to the carrier (the ratio is 1'540). The maximal absolute Doppler on the code can thus be estimated as 3.2 Hz.

### GPS data message

The GPS message is a continuous data stream providing the receiver the information necessary for the PVT solution computation. Each satellite relays the following information:

- system time and clock correction values;
- its own exact orbital data (ephemeris);
- other satellites approximated orbital data (almanac);
- system health, etc.

The navigation message consists in 25 frames (or pages) of 1500 bits each, transmitted at the rate of 50 bps in 12.5 minutes. Each frame is distributed into five subframes of 300 bits that are in turn divided into 10 words of 30 bits. The structure of the complete navigation message is represented in Figure 2.2.

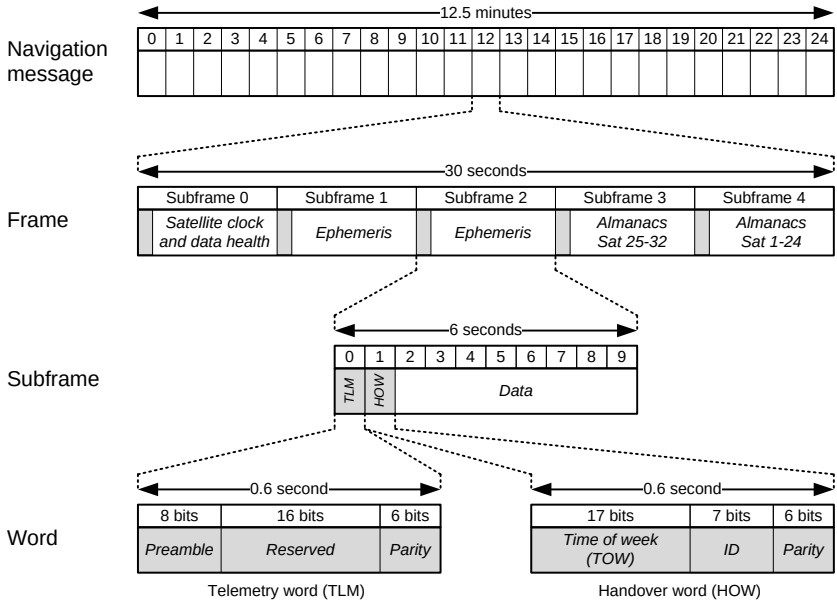


**Figure 2.2:** Structure of the GPS L1 navigation message [Zog09].

The first word of every subframe is the Telemetry word (TLM), used for synchronization purpose. It is followed by the Handover Word (HOW), containing the Time Of Week (TOW), which counts the time elapsed since the beginning of the GPS week (every Sunday at 00h00), by increment of 6 seconds. The eight remaining data words are specific to each subframe.

The first subframe contains the clock information and the health of the broadcasting satellite. It also transmits the week number, counting the number of weeks elapsed since the beginning of the GPS time (Sunday $6^{th}$ January 1980 at 00h00). Subframes 2 and 3 hold the ephemeris, or exact orbital parameters, specific to the satellite. The first three subframes are identical for all the 25 frames and therefore the most important data for the position determination are transmitted every 30 seconds. The subframes 4 and 5 contain the almanacs, or approximated orbital characteristics, of the whole constellation. Each satellite broadcasts the almanac data for all the satellites, but only its own ephemeris data. A more detailed analysis of the message structure is provided in [Tsu05].

A receiver must have collected the complete ephemeris of at least four satellites in order to be able to compute its position. Depending on the information a priori known and their actuality, the procedure can require more or less time, and can be classified as follows [ike10]:

- *cold start*: no a priori information about the satellites is known. This happens when the receiver was switched off for a long time or has moved a few hundreds kilometers away from its last position fix;
- *warm start*: the time of the receiver and the almanac data are known, but the ephemeris data are out of date. This happens when more than 3–6 hours have elapsed since the last position fix;
- *hot start*: the time of the receiver and the almanac data are known and the ephemeris data are up-to-date. This happens when the receiver is turned on at approximately the same position within 2 hours after the last position fix.

### GPS identification codes

The identification codes used in GPS consist in binary sequences that appear to be random with noise like properties, but which are actually deterministic. Because of these characteristics, they are referred as Pseudo Random Noise (PRN) codes.

Since all the satellites broadcast on the same L1 and L2 carrier frequencies, the signals origin is distinguished by assigning each satellite a unique PRN code. This technique of data multiplexing is known as Code Division Multiple Access (CDMA) [Raz98]. The receiver can then identify and decode a specific satellite data by correlating (i.e. multiplying and accumulating) the incoming signal with a locally generated replica of the code. The PRN codes have excellent autocorrelation properties and are highly mutually orthogonal, so that it is unlikely that one satellite signal will be misinterpreted as another under normal signal conditions [Sar80].

**GPS CA code** The CA code, also known as *Gold code* as Robert Gold described it in 1967 [Gol67], consists in a 1023 chips sequence (the name "chip" is used instead of "bit" as no information is carried by the code). It is generated at the rate of 1.023 MHz which corresponds to an equivalent main lobe null-to-null bandwidth of 2.046 MHz. Each code epoch lasts one millisecond and repeats itself every millisecond. Every CA code sequence is constructed by combining different outputs of two 10-cell shift registers with proper feedback and the satellite identification is determined by the two output positions of the second register [Kap06].

Two fundamental properties of the GPS L1 CA code can be expressed as follows [Bor07]:

1. *Nearly no cross-correlation:* all the CA codes are nearly uncorrelated with each other. That is, for two codes $C_i$ and $C_j$ of the satellites $i$ and $j$, the cross-correlation function $R_{ij}(m)$ can be expressed as:

$$R_{ij}(m) = \sum_{k=0}^{1022} C_i(k) \cdot C_j(k+m) \approx 0 \qquad \text{for all } m \tag{2.7}$$

2. *Nearly no correlation except for zero lag:* all the CA codes are nearly uncorrelated with themselves, except for zero lag. That is, for the code $C_i$ of the satellite $i$, the autocorrelation function $R_{ii}(m)$ can be expressed as:

$$\begin{aligned} R_{ii}(0) &= \sum_{k=0}^{1022} C_i(k) \cdot C_i(k) = 1023 \\ R_{ii}(m) &= \sum_{k=0}^{1022} C_i(k) \cdot C_i(k+m) \approx 0 \qquad \text{for } m \neq 0 \end{aligned} \tag{2.8}$$

An example of the autocorrelation function of the GPS L1 CA code is illustrated in Figure 2.3.

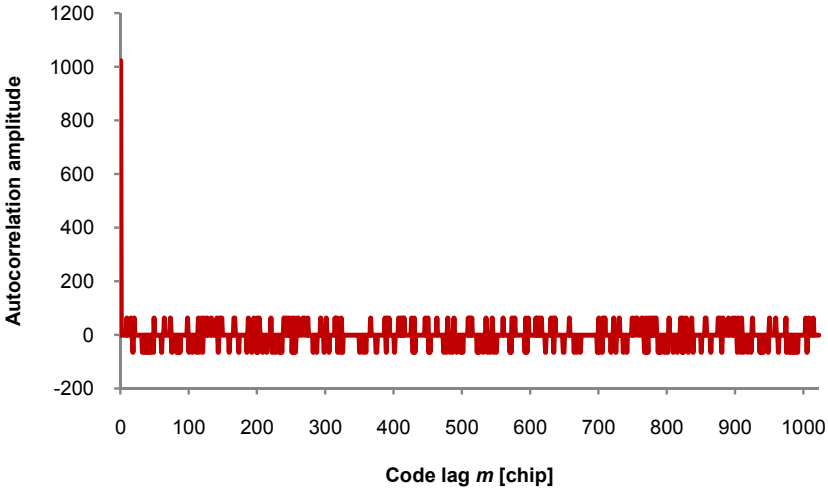


**Figure 2.3:** Autocorrelation function $R_{11}(m)$ of the GPS L1 CA code. $R_{11}(0) = 1023$ while $|R_{11}(m \neq 0)| \leq 65$.

**GPS P(Y) code** The P(Y) code consists in a sequence of approximately $2.3 \cdot 10^{14}$ chips, generated at the rate of 10.23 MHz, which corresponds to an equivalent main lobe null-to-null bandwidth of 20.46 MHz. The code is constructed by combining two original PRN sequences of respectively 15'345'000 and 15'345'037 chips. Thus, the P(Y) code period theoretically lasts more than 38 weeks, but is currently reset and repeated every week [Kap06]. The actual P(Y) code is not directly transmitted, but is encrypted by an auxiliary Y sequence. The resulting code is mainly used for military purpose and is not available to civilian users.

Table 2.1 summarizes the main signals characteristics of the GPS L1 and L2 signals.

| | GPS L1 | GPS L2 |
|---|---|---|
| Carrier frequency | 1'575.42 MHz | 1'227.6 MHz |
| Code rate | 1.023 MHz (CA)<br>10.23 MHz (P(Y)) | 10.23 MHz (P(Y)) |
| Code length | 1023 chips (CA)<br>$2.3 \cdot 10^{14}$ chips (P(Y)) | $2.3 \cdot 10^{14}$ chips (P(Y)) |
| Data rate | 50 bps | |

**Table 2.1:** Characteristics of the GPS L1 and L2 signals.

### 2.1.2 GPS L1 CA signal modulation

This section focuses now on the L1 band and the CA code in particular, since it is public and used by all the current civilian GPS receivers. All the GPS satellites are equipped with four extremely stable atomic clocks producing a frequency of 10.23 MHz from which are derived the carrier frequency, the data message and the identification codes. The GPS L1 CA signal is obtained by first combining the low bit-rate data message (50 bps) with the high CA code frequency $f_c = 1.203$ MHz, spreading the data spectrum over an approximately 2 MHz bandwidth. Assuming that both the binary sequences are represented with 1's and -1's values, the latter operation translates into a simple multiplication. The resulting sequence then modulates the carrier signals L1 using the Binary Phase Shift Keying (BPSK) scheme, where each data transition affects the carrier frequency by a 180° phase shift. An example of BPSK modulation is shown in Figure 2.4.

The signal broadcasted by each satellite in the GPS L1 CA band can be represented as follows [Die95]:

$$s(t) = a_t \cdot d(t) \cdot c(t) \cdot \cos(\omega_{L1} \cdot t + \phi_{L1}) \tag{2.9}$$

where $a_t$ is the signal amplitude;
$d(t) = \pm 1$ is the data message, specific to each satellite;
$c(t) = \pm 1$ is the CA code sequence, specific to each satellite;
$\omega_{L1} = 2 \cdot \pi \cdot f_{L1}$ is the L1 carrier angular frequency [Hz];
$\phi_{L1}$ is the L1 carrier phase [rad].

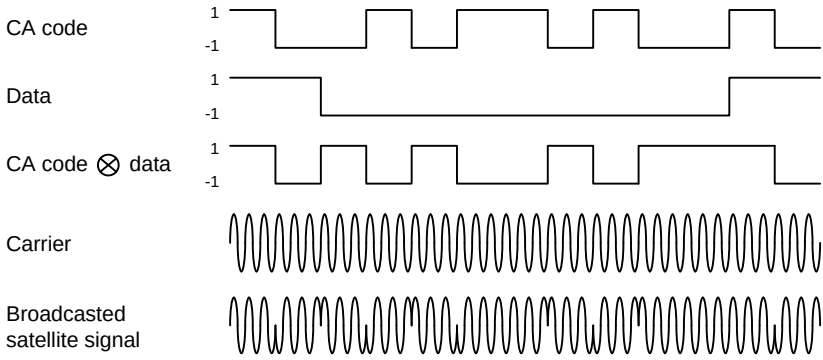


**Figure 2.4:** Example of BPSK modulation [Bor07].

A block diagram of the GPS L1 CA signal generation is given in Figure 2.5.

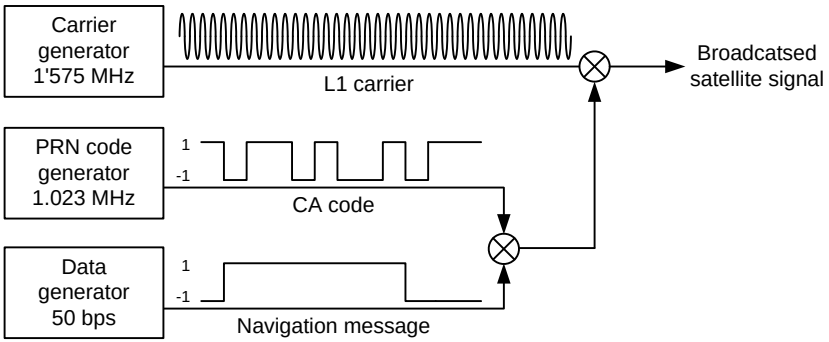


**Figure 2.5:** Generation of the GPS L1 CA signal [Zog09].

According to the standard positioning service signal specification [US 95], the minimal signal power received on earth (in open sky conditions) is between -160 and -158 dBW, depending on the satellite elevation angle. Considering the approximately 2 MHz null-to-null bandwidth of the CA code signal and a receiver temperature $T = 290$ degrees K, the signal power is well below the thermal noise floor defined as:

$$P_n = 10 \cdot log_{10}(k \cdot T \cdot B_n) \approx -141 \text{ dBW} \tag{2.10}$$

where $k \approx 1.38 \cdot 10^{-23}$ J/K is the Boltzmann constant;
$B_n$ is the equivalent noise bandwidth of the signal [Hz].

Thus, under normal operating conditions, the Signal-to-Noise Ratio (SNR) at the receiver input is between -19 and -17 dB. The signal is dominated by noise and requires further appropriate processing to be detected.

### 2.1.3 Galileo E1 signals

Galileo is a GNSS system currently developed by the European Union in cooperation with the European Space Agency. It will rely on a constellation of 30 satellites orbiting on three circular plans at an altitude of 23'222 km above the earth surface. The system will provide several navigation services, broadcasted on different frequencies with different modulation schemes [Eur10b]:

- the Open Service (OS) will be free for anyone to access and will be broadcasted in two frequency bands, centered at 1'191.795 and at 1'575.42 MHz. The receivers processing the two bands should achieve a horizontal accuracy <4 m, while the receivers that use only a single band will achieve an accuracy of <15 m, comparable to what the civilian GPS L1 CA service provides today [Zog09]. It is expected that most of the future mass market receivers will process both the GPS L1 CA and the Galileo OS signals for obtaining a maximum of coverage;

- the encrypted Commercial Service (CS) will be available for a fee. It will offer a higher data throughput rate and enable the users to improve the accuracy. It will be broadcasted in three frequency bands, the two used for the OS signals plus at 1'278.75 MHz;

- the encrypted Public Regulated Service (PRS) and Safety of Life Service (SoL) will both achieve an accuracy comparable to the OS, but with an improved robustness against jamming and faster and more reliable detection of problems. They target security authorities (police, military, etc.) and safety-critical transport applications (air-traffic control, automated aircraft landing, etc.).

As compared to the BPSK modulation, the Galileo signals are characterized by the introduction of a square wave sub-carrier modulation known as Binary Offset Carrier (m,n) (BOC), where $m$ stands for the ratio between the sub-carrier frequency and the reference frequency $f_0 = 1.023$ MHz, and $n$ stands for the ratio between the code rate $f_c$ and $f_0$. The goal of the BOC modulation is to better distribute the signal over the bandwidth in order to improve the immunity to thermal noise and to multipath errors.

It also helps minimizing the mutual impact when used simultaneously with another modulation such as the BPSK, since their power spectrum density maxima are separated [Mar03]. This is illustrated in Figure 2.6.

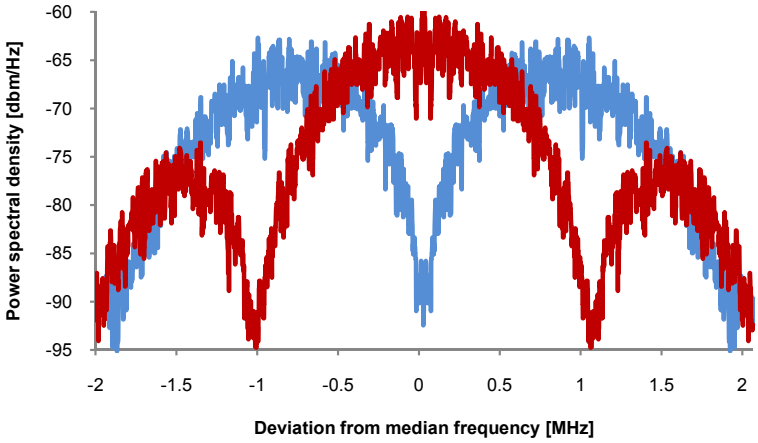


**Figure 2.6:** Comparison between the power spectral density of the BOC(1,1) (blue curve) and the BPSK (red curve) modulations. The signals strength is normalized to 1 W [Zog09].

However, the drawback of the BOC modulation is the ambiguity of the autocorrelation function caused by the introduction of additional lobes. This increases the risk of false peak detection and makes the acquisition process more complex [Wae07]. Figure 2.7 compares the GPS L1 CA autocorrelation function with the BOC(1,1) one, that presents two secondary negative lobes disposed around the main peak.

The rest of this section focuses on the freely accessible Galileo E1 OS signal, transmitted on the 1'575.42 MHz carrier frequency. It is composed of two channels, the data signal E1-B and the dataless signal E1-C, also called pilot and made of an identification code only [Eur10a]. Both are based on a combination of two BOC modulations (referenced as Composite BOC (CBOC)) added together with different weighting, as follows [Zog09]:

$$\mathrm{CBOC}_{E1} = \frac{10}{11} \cdot \mathrm{BOC}(1,1) + \frac{1}{11} \cdot \mathrm{BOC}(6,1) \tag{2.11}$$

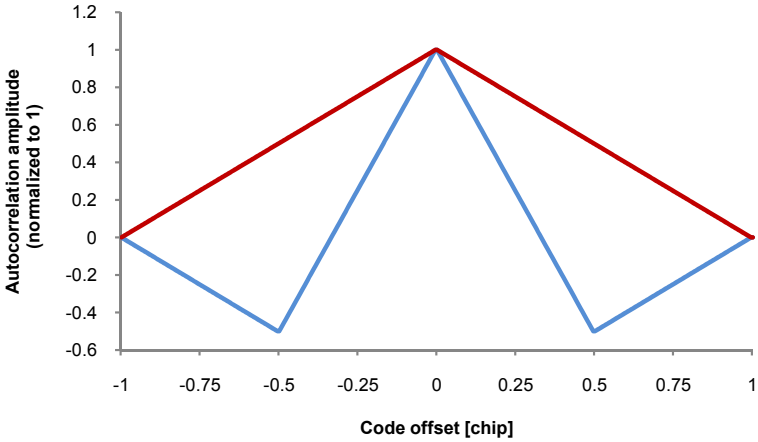


**Figure 2.7:** Comparison between the Galileo E1-B BOC(1,1) (blue curve) and the GPS L1 CA (red curve) autocorrelation functions.

The Galileo E1 OS signal has a code length of 4092 chips generated at the rate of 1.023 MHz. Each code epoch lasts 4 milliseconds and repeats itself. For the pilot channel, a secondary code of 25 chips extends the repetition interval to 100 milliseconds [Eur10a]. The main signal characteristics are regrouped in Table 2.2.

| | Galileo E1-B | Galileo E1-C |
|---|---|---|
| Carrier frequency | 1.575.42 MHz | |
| BOC modulation | CBOC | |
| Code rate | 1.023 MHz | |
| Sub-carrier rate | 1.023 MHz + 6.138 MHz | |
| Code length | 4092 chips | 4092 x 25 chips |
| Data rate | 250 bps | - |

**Table 2.2:** Characteristics of the Galileo E1 signal. The E1-C channel uses both a primary code of 4092 chips modulated by a secondary one of 25 chips, resulting in a 4092 x 25 code length and called tiered code.

## 2.2 Receiver RF front-end

The RF front-end constitutes the first element of the processing chain. It is responsible for receiving, conditioning and digitizing the incoming signal in order to provide the microprocessor with a proper digital data stream. The satellites signals traveling through space are first captured by the receiver antenna that induces a voltage which amplitude is too weak and its frequency too high to be directly processed by most ADCs. Therefore, the front-end utilizes a combination of amplifier(s), filter(s), and mixer(s) in order to properly condition the signal prior to digitization. An example of the processing chain of a RF front-end is illustrated in Figure 2.8.

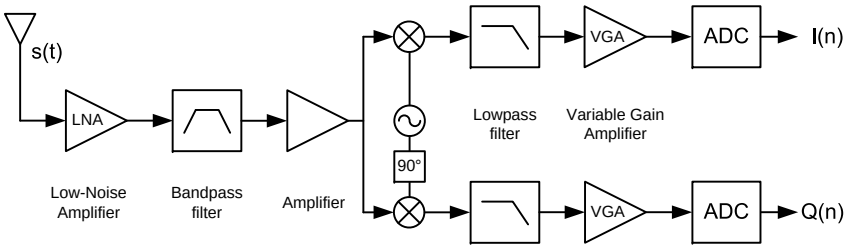


**Figure 2.8:** Example of a typical RF front-end topology.

The first component after the antenna is generally a Low-Noise Amplifier (LNA). Since the requested gain may be higher than 100 dB to fit closely the ADC voltage input range (typically of a few hundreds millivolt), several amplifiers are usually used and distributed along the front-end [Tsu05]. The amplified signal then goes through a bandpass filter centered at the carrier frequency in order to attenuate the out-of-band RF interferences entering the rest of the processing chain. If the total gain is too high, the output signal will be saturated and if too low, the dynamic range of the ADC will not be fully exploited. For this reason, the last amplification stage is usually a Variable Gain Amplifier (VGA).

The signal is then down-converted from the RF to a lower Intermediate Frequency (IF), generally of a few megahertz, which is more suitable to operate the upcoming analog-to-digital conversion. The frequency translation is performed in a mixer that multiplies the signal with a local complex carrier (i.e. sine and cosine) that is derived from the reference oscillator.

The process operates through the following trigonometric identities:

$$\begin{aligned}
\cos(\omega_1 \cdot t) \cdot \cos(\omega_2 \cdot t) &= \frac{1}{2} \cdot [\cos((\omega_1 - \omega_2) \cdot t) + \cos((\omega_1 + \omega_2) \cdot t)] \\
\sin(\omega_1 \cdot t) \cdot \sin(\omega_2 \cdot t) &= \frac{1}{2} \cdot [\cos((\omega_1 - \omega_2) \cdot t) - \cos((\omega_1 + \omega_2) \cdot t)] \\
\sin(\omega_1 \cdot t) \cdot \cos(\omega_2 \cdot t) &= \frac{1}{2} \cdot [\sin((\omega_1 - \omega_2) \cdot t) + \sin((\omega_1 + \omega_2) \cdot t)]
\end{aligned} \tag{2.12}$$

The mixing process generates both a lower and an upper signal component, the latter being rejected by a post mixer lowpass (or bandpass) filter. It also preserves the signal properties (Doppler and PRN code) since only the carrier frequency is lowered. The mixer down-converts the input signal into two sets of real and imaginary data, generally denoted as $I(t)$ and $Q(t)$. Depending on the architecture, some front-ends may only output one single real data stream $I(t)$.

The final component of the front-end is the ADC which is responsible for the analog signal conversion into digital samples at the frequency $f_s$. The ADC is generally preceded by an anti-aliasing bandpass filter around the CA code that limits the band to the frequencies of interest. Because of the narrower bandwidth of the CA code, the P(Y) code is filtered out, distorted and cannot be demodulated [Bor07]. At the front-end output, the GPS L1 CA signal contribution from each satellite can be mathematically expressed as:

$$\begin{aligned}
I(n) &= a_n \cdot d(n) \cdot c(n) \cdot \cos(\omega \cdot n \cdot T_s + \phi) \\
Q(n) &= a_n \cdot d(n) \cdot c(n) \cdot \sin(\omega \cdot n \cdot T_s + \phi)
\end{aligned} \tag{2.13}$$

where $a_n$ is the signal amplitude;
$n$ is a positive integer denoting the $n^{th}$ sample;
$n \cdot T_s = n/f_s$ is the sampling time [s];
$\omega$ is the carrier angular frequency (plus Doppler) [Hz];
$\phi$ is the carrier phase [rad].

Depending on the front-end components specifications or the design trade-offs, multiple stages of amplification, filtering and frequency translation may be required to obtain the desired output IF. However, this may be not desirable for a software receiver, where the objective is to place the ADC as close as possible to the antenna for minimizing the components. In that sense, new approaches exploiting a simplified front-end topology have been proposed in the literature and are described hereafter [Ako96].

### 2.2.1 Direct sampling

The minimal front-end design is obtained by sampling the incoming RF signal directly at the carrier frequency, without any prior down-conversion. Accordingly to the Nyquist theorem, the lowest sampling rate must be at least twice the highest signal frequency component. Considering the GPS L1 carrier frequency $f_{L1}$ and the CA code bandwidth $B_c$, this translates into the following requirement:

$$f_s > 2 \cdot \left( f_{L1} + \frac{B_c}{2} \right) \approx 2 \cdot 1.576 \text{ GHz} = 3.152 \text{ GHz} \tag{2.14}$$

Ideally the digitization would take place right next to the antenna, but proper signal amplification and filtering is still required in order to fit the input range of the ADC and avoid aliasing. Consequently, the direct sampling technique requires a state-of-the-art amplifier and an ADC operating at a frequency higher than 3 GHz, as well as a bandpass filter centered at the RF carrier frequency. The fabrication of components operating at such high frequencies remains critical and their integration in the front-end design may drastically impact the cost of the receiver. But, most of all, the direct sampling results in a prohibitive data rate of at least 3 Gbps that cannot be handled and further processed by a microprocessor. Therefore, this type of implementation is impractical for the time being and alternative approaches must be considered.

### 2.2.2 Bandpass sampling

The bandpass sampling technique exploits the information bandwidth of the signal rather than the carrier frequency. The idea is that a bandpass signal can be translated to a lower frequency band if sampled at a rate greater than twice its bandwidth. Considering the null-to-null bandwidth $B_c$ of the GPS L1 CA code, the constraint of Equation 2.14 is relaxed and the minimal frequency becomes:

$$f_s > 2 \cdot B_c = 2 \cdot 2.046 \text{ MHz} = 4.092 \text{ MHz} \tag{2.15}$$

Consequently, the incoming RF signal is intentionally undersampled to achieve frequency translation via aliasing [Ako99]. If the sampling frequency is properly chosen, the process folds the entire information bandwidth, together with noise, into the resulting sampled band of $[0; f_s/2]$ Hz without generating interferences. The principle is illustrated in Figure 2.9.

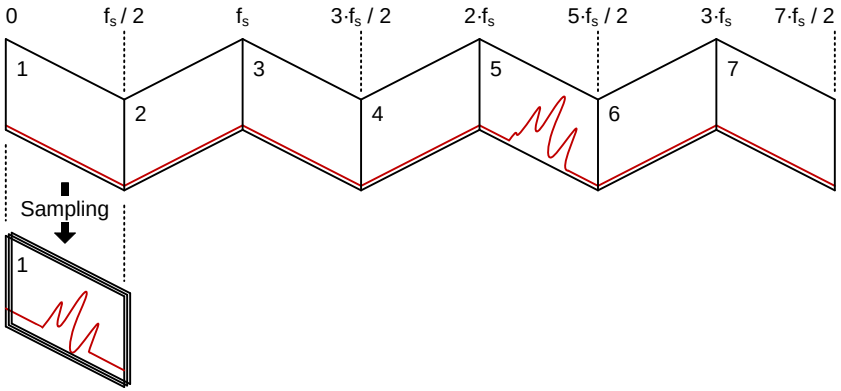


**Figure 2.9:** Fan-fold paper showing the spectrum of a bandpass signal. After sampling, all of the sheets superimposed on top of each other and fold into the band between 0 and $f_s/2$ Hz [Pen10].

By varying the sampling frequency, the aliasing components of the discrete spectrum shift left and right, up to a point where they overlap. The criterion ensuring that no part of the signal information bandwidth $B_c$ will fold on the top of itself can be expressed as [Tse02]:

$$\frac{2 \cdot f_{L1} + B_c}{m} \leq f_s \leq \frac{2 \cdot f_{L1} - B_c}{m-1} \text{ [Hz]} \tag{2.16}$$

where $m$ is an arbitrary positive integer in the range $1 \leq m \leq \frac{2 \cdot f_{L1} + B_c}{2 \cdot B_c}$

For $m = 1$, the above equation states the Nyquist sampling criterion of Equation 2.14. For $m > 1$, $f_s$ becomes smaller than the Nyquist frequency and subsampling is operated.

The incoming 1'575 GHz carrier frequency can thus be undersampled at a few megahertz, relaxing the ADC speed requirements and the resulting data rate. However, the bandpass sampling still imposes severe constraints on the hardware components. First, both the amplifier and the ADC inputs must accommodate the L1 carrier frequency. Second, a very narrow bandpass filter centered at the RF carrier frequency is required, as all the frequencies ranging from 0 Hz to the input bandwidth of the ADC will be aliased and fold into the band of interest, thus degrading the SNR.

Considering the GPS L1 CA code bandwidth, the required quality factor $Q_f$ of the filter is expressed as:

$$Q_f = \frac{f_{L1}}{B_c} = \frac{1'575 \text{ MHz}}{2.046 \text{ MHz}} \approx 770 \tag{2.17}$$

This value is extremely high as compared to commercial filters with a quality factor typically of 50 [Bor07]. Consequently, to relax the constraints on the filter bandwidth and overcome the issue of related noise aliasing, much higher sampling frequencies have to be used. Here lies the first compromise of a software receiver, where the simplification of the front-end design negatively impacts both the cost and the complexity of the system. Bandpass sampling may be the answer to achieve digitization closer to the antenna, but too many constraints still limit its realization as compared to traditional IF translation[1], which provides higher flexibility.

## 2.3 Receiver base-band processing

The base-band processing is the core of the receiver and aims to down-convert the RF front-end output signal to base-band (i.e. where the signal spectrum is centered at 0 Hz) in order to extract the information necessary to the PVT solution computation. The traditional hardware receivers are organized in a parallel structure where the incoming signal is distributed to the multiple satellite channels (typically 12) responsible for the simultaneous processing of the different satellites. The typical channelized structure of a receiver is illustrated in Figure 2.10.

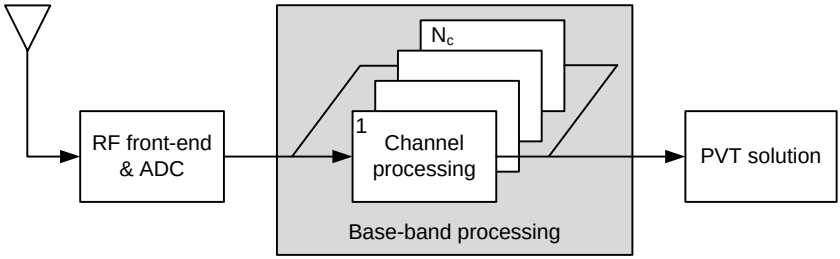


**Figure 2.10:** Channelized organization of the base-band processing.

[1]Bandpass sampling may also be applied at IF, which allows to suppress the last down-conversion stage. However this does not go toward the initially targeted RF front-end design simplification.

The operations are identical for all the satellite channels and consist in locally generating a local replica of the carrier and code for sequentially removing the residual carrier (IF plus Doppler) and the CA code from the incoming signal, and finally accumulating the resulting samples over a predetermined period. Since all these operations are performed at the system frequency rate and concurrently for several satellites, the base-band processing is traditionally implemented in an ASIC to ensure the real-time operation of the receiver.

### 2.3.1 Carrier and code generation

The base-band demodulation requires the local generation of the complex carrier (for the carrier down-conversion) and code replicas (for the code removal). In traditional hardware receivers, both the carrier and code waveforms are produced by means of a Numerically Controlled Oscillator (NCO), which emulates a digital frequency generator by increasing a phase accumulator with a phase increment on a per sample basis. The accumulated phase value is then looked up in a table and converted into the corresponding amplitude in order to recreate the desired waveform sampled at the system frequency [Kap06]. One entire cycle is completed each time the NCO accumulator overflows, as illustrated in Figure 2.11. The generated frequency $f$ is proportional to the sampling frequency $f_s$ and the phase increment $Inc$:

$$f = \frac{f_s \cdot Inc}{2^W} \text{ [Hz]} \tag{2.18}$$

where $W$ is the accumulator bit width.

The NCO frequency resolution $\delta f$ is given by:

$$\delta f = \frac{f_s}{2^W} \text{ [Hz]} \tag{2.19}$$

For a 32-bit accumulator, the resolution of a NCO operating at a few megahertz is in the range of a few millihertz.

As the incoming satellites signal is quantized with $M$ bits, it is usually sufficient to represent the values taken by the local carrier as $M$-bit integer as well, reducing the frequency waveform to $2^{M+1}$ phases that assume $2^M$ different values. Three bits are enough for encoding the sine and cosine waveforms with good performances, as no more than 0.02 dB loss can be expected during the Doppler removal process [Die95].

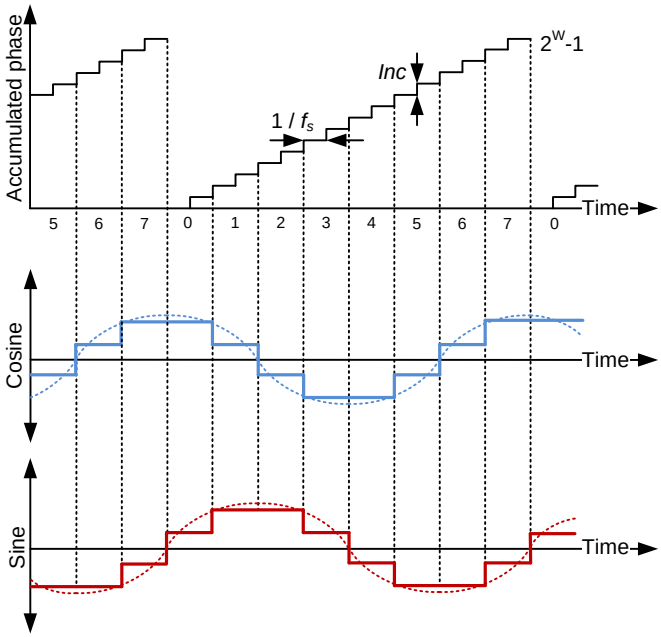


**Figure 2.11:** NCO staircase output converted into a complex waveform.

Table 2.3 regroups the different magnitude levels of a 3-bit quantized complex carrier, while Figure 2.12 compares the original sine wave with the 3-bit quantized one.

| Carrier phase | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Sine | 1 | 3 | 4 | 5 | 5 | 4 | 3 | 1 |
| Cosine | 5 | 4 | 3 | 1 | -1 | -3 | -4 | -5 |
| Carrier phase | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Sine | -1 | -3 | -4 | -5 | -5 | -4 | -3 | -1 |
| Cosine | -5 | -4 | -3 | -1 | 1 | 3 | 4 | 5 |

**Table 2.3:** 3-bit sine and cosine quantization values.

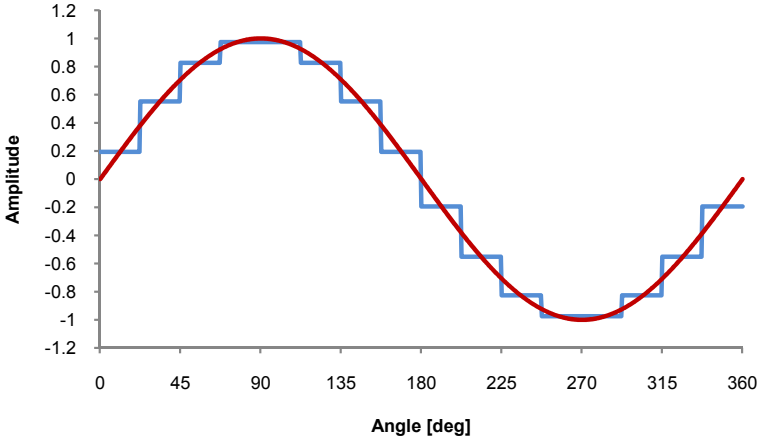


**Figure 2.12:** 3-bit quantized sine wave (blue curve) versus original continuous sine wave (red curve).

The code replicas are synthesized in the same way, directly from the phase of the code NCO, but without generating the sine and cosine components. The same clock and frequency properties described above also apply.

### 2.3.2 Base-band demodulation

For the sake of simplicity, the description of the demodulation is given here for one satellite only. The residual carrier $\omega$ is first wiped-off by multiplying the incoming signal samples with the local complex carrier wave, synthesized by the carrier generator at the estimated frequency $\widetilde{\omega}$ and phase $\widetilde{\phi}$. If the RF front-end outputs complex-valued IF samples (i.e. both $I(n)$ and $Q(n)$), the mixing is implemented as [Die95]:

$$\begin{aligned} I_{bb}(n) &= I(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) + Q(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \\ Q_{bb}(n) &= Q(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) - I(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \end{aligned} \quad (2.20)$$

The frequency translation operates through Equation 2.12 and the mixer outputs two separate in-phase and quadrature sequences $I_{bb}(n)$ and $Q_{bb}(n)$ in the base-band.

The result of the down-conversion can be expressed as:

$$\begin{aligned} I_{bb}(n) &= a_n \cdot d(n) \cdot c(n) \cdot \cos(\Delta\omega \cdot n \cdot T_s + \Delta\phi) \\ Q_{bb}(n) &= a_n \cdot d(n) \cdot c(n) \cdot \sin(\Delta\omega \cdot n \cdot T_s + \Delta\phi) \end{aligned} \tag{2.21}$$

where $\Delta\omega = \omega - \widetilde{\omega}$ is the angular frequency mismatch [Hz];
$\Delta\phi = \phi - \widetilde{\phi}$ is the phase mismatch [rad].

Note that for real-valued IF input samples (i.e. $I(n)$ only), the mixer generates both a lower and an upper sideband component, adding an extra term in Equation 2.21 at approximately twice the original IF signal:

$$\begin{aligned} I_{bb}(n) = &\frac{a_n}{2} \cdot d(n) \cdot c(n) \cdot \cos(\Delta\omega \cdot n \cdot T_s + \Delta\phi) \\ &+ \frac{a_n}{2} \cdot d(n) \cdot c(n) \cdot \cos((\omega + \widetilde{\omega}) \cdot n \cdot T_s + \phi + \widetilde{\phi}) \\ Q_{bb}(n) = &\frac{a_n}{2} \cdot d(n) \cdot c(n) \cdot \sin(\Delta\omega \cdot n \cdot T_s + \Delta\phi) \\ &- \frac{a_n}{2} \cdot d(n) \cdot c(n) \cdot \sin((\omega + \widetilde{\omega}) \cdot n \cdot T_s + \phi + \widetilde{\phi}) \end{aligned} \tag{2.22}$$

However, the influence of this extra high frequency component is minimized through the upcoming correlation process acting as a lowpass filter, as explained hereafter.

The next step is to remove the CA code from the two signals $I_{bb}(n)$ and $Q_{bb}(n)$, by correlating them with a local code replica. To ensure further proper tracking operations, three different time delayed replicas of the code, denoted as $c^E(n)$, $c^P(n)$, and $c^L(n)$ are used. The indexes Early (E), Prompt (P), and Late (L) refer to the time shift of each replica with respect to the incoming signal code phase, $c^P(n)$ being aligned and producing the maximum correlation, $c^E(n)$ and $c^L(n)$ being respectively shifted a fraction of chip earlier and later, producing part of the maximum correlation. $I_{bb}(n)$ and $Q_{bb}(n)$ are thus multiplied point by point with these three code replicas to produce six signal components $I^E(n)$, $I^P(n)$, and $I^L(n)$, and $Q^E(n)$, $Q^P(n)$ and $Q^L(n)$. The latter are separately accumulated over the integration period $T_{int}$ to produce three in-phase and quadrature correlator outputs denoted as $I^E$, $I^P$, and $I^L$, and $Q^E$, $Q^P$ and $Q^L$.

$$I^x = \sum_{n=0}^{N_s-1} I_{bb}(n) \cdot c^x(n) = \sum_{n=0}^{N_s-1} I^x(n)$$
$$Q^x = \sum_{n=0}^{N_s-1} Q_{bb}(n) \cdot c^x(n) = \sum_{n=0}^{N_s-1} Q^x(n) \tag{2.23}$$

where $x \in \{E, P, L\}$;
$N_s = f_s \cdot T_{int}$ is the number of samples accumulated over $T_{int}$.

The correlation operation de-spreads the signals $I_{bb}(n)$ and $Q_{bb}(n)$ and achieves proper gain in order to recover them from the noise floor. The process acts as a low pass filter which bandwidth $B$ is inversely proportional to the integration period $T_{int}$:

$$B = \frac{1}{T_{int}} \text{ [Hz]} \tag{2.24}$$

By coherently processing 1 ms of data, corresponding to one complete CA code period, the equivalent bandwidth of the signal becomes 1 kHz. However, this also affects the equivalent noise bandwidth in Equation 2.10 and the thermal noise power becomes $P_n = -174$ dBW for $B_n = 1$ kHz, so producing a SNR of 14 dB (-160 + 174 dB) at the correlator output [Tsu05]. More generally, increasing the coherent integration time by a factor $m$ with respect to 1 ms provides an additional correlation gain $G$ of:

$$G = 10 \cdot \log_{10}(m) \tag{2.25}$$

For example, doubling the integration time to 2 ms leads to a 3 dB increase of the SNR at the correlator output, and so on. The length of the integration time is theoretically limited by the navigation data occurring every 20 ms. However, when knowing the value of the transmitted data bits, this time can be additionally increased to achieve higher sensitivity [Zog09].

Under the assumption that no data bit transition occurs during the integration interval (i.e. $d$ remains constant), Equation 2.23 can be approximated by [Die95]:

$$\begin{aligned} I^x &\approx a_n \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot \Delta f \cdot T_{int})}{\pi \cdot \Delta f \cdot T_{int}} \cdot \cos(\pi \cdot \Delta f \cdot T_{int} + \Delta\phi) \\ Q^x &\approx a_n \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot \Delta f \cdot T_{int})}{\pi \cdot \Delta f \cdot T_{int}} \cdot \sin(\pi \cdot \Delta f \cdot T_{int} + \Delta\phi) \end{aligned} \tag{2.26}$$

where $R^x(\Delta\tau)$ is the autocorrelation function of the CA code defined as:

$$R^x(\Delta\tau) = \begin{cases} 1 - |\Delta\tau| & \text{for } |\Delta\tau| < 1 \text{ chip;} \\ \approx 0 & \text{for } |\Delta\tau| \geq 1 \text{ chip.} \end{cases} \tag{2.27}$$

The implementation of the above base-band demodulation results in the two generic satellite channel architectures, respectively with real and complex input data, illustrated in Figure 2.13. If both the code and the carrier are perfectly replicated (i.e. $\Delta\tau = 0$, $\Delta f = 0$, and $\Delta\phi = 0$), the magnitudes of the in-phase correlator outputs are maximum and the energy equally distributed between the E and L components. On the other hand, any code misalignment introduces a correlation loss and produces an imbalance between the E and L correlator outputs. In the same way, if a carrier mismatch subsists, the in-phase and quadrature correlation results are time modulated by a residual frequency and their amplitude affected by an additional loss.

After each integration, all the correlation results are dumped and transmitted to the base-band algorithms in order to extract the information necessary to the receiver operations. When the receiver stands in acquisition and searches for a satellite, these values are used to determine the presence or not of the signal, while in tracking they are used for refining the carrier and code parameters estimation.

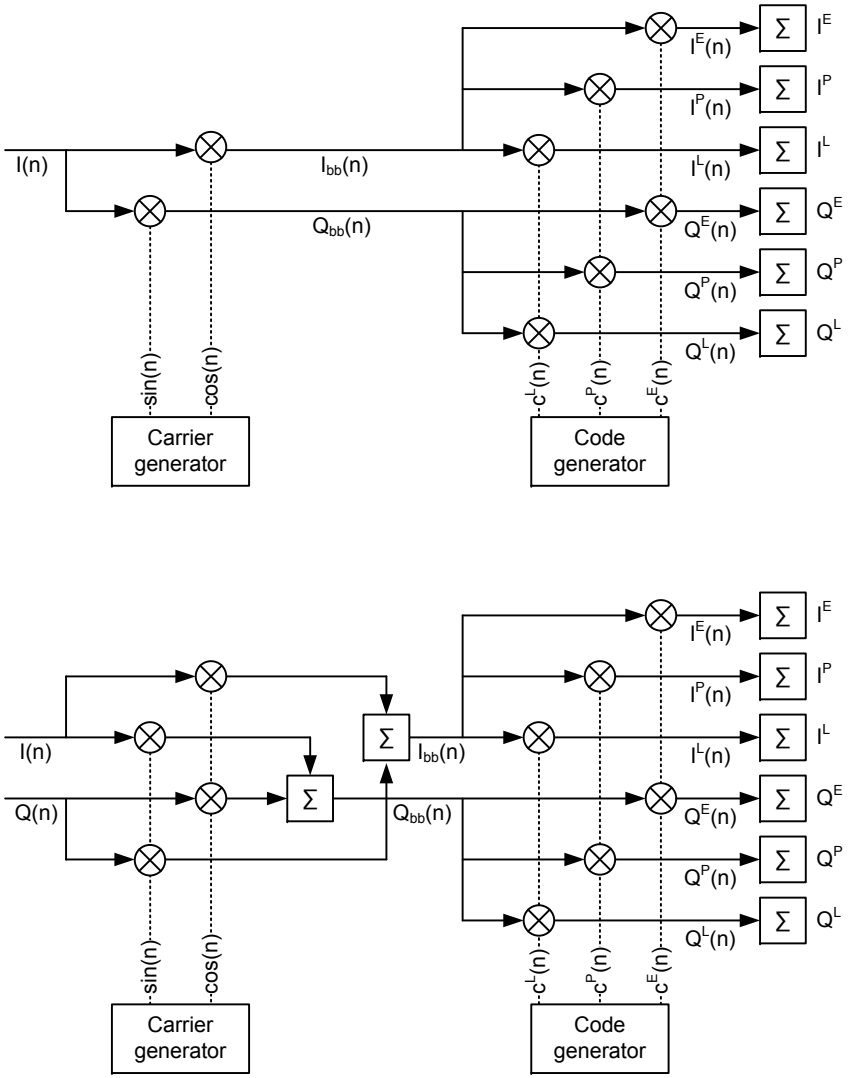


**Figure 2.13:** Real and complex generic base-band architectures with respectively carrier removal, code removal and accumulation stages. Only one single satellite channel is illustrated.

## 2.4 Receiver base-band algorithms

The base-band algorithms are executed only once per integration, at a much lower rate than the base-band operations. They are often carried out by a small microprocessor (as well as the PVT solution computation), generally directly integrated within the ASIC.

### 2.4.1 Acquisition algorithms

After powering up the receiver, each satellite channel is attributed a satellite to search for and starts replicating and testing all the possible phases of its local code replica until it correlates with the incoming signal. The receiver must also detect the satellite in the carrier frequency dimension in order to compensate for the Doppler effect due to the relative motion between the satellite and the receiver. The Doppler frequency range is thus swept with a constant step size (also called bin width) until the carrier matching is achieved. Consequently, the acquisition consists in a two-dimensional search process with the uncertainty ranging from [0; 1023] chips for the code and from [-5; 5] kHz for the carrier frequency, as illustrated in Figure 2.14.

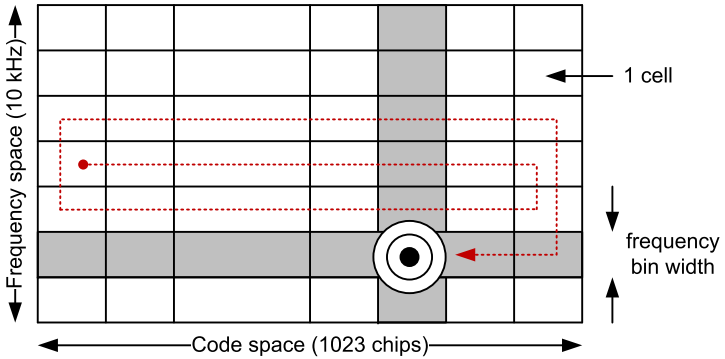


**Figure 2.14:** Two-dimensional code and frequency search space.

For each tested code and frequency combination (also called cell), the envelope $Env$ of the signal is computed as follows:

$$Env = \sqrt{I^2 + Q^2} = a \cdot N_s \cdot R(\Delta\tau) \cdot \left| \left[ \frac{\sin(\pi \cdot \Delta f \cdot T_{int})}{\pi \cdot \Delta f \cdot T_{int}} \right] \right| \qquad (2.28)$$

If the signal envelope is higher than a pre-defined threshold, the satellite is declared as present and the satellite channel switches to tracking with the estimated code phase and carrier frequency. Otherwise it keeps testing the remaining cells until the whole uncertainty space is swept.

The smaller the search step sizes of the code and frequency, the higher the sensitivity, but also the complexity (i.e. the number of cells to test) and thus, the time of the acquisition. Table 2.4 regroups different code step size configurations with their impact on the search sensitivity.

| | Code step size | | |
|---|---|---|---|
| | $\frac{1}{2}$ chip | $\frac{1}{4}$ chip | $\frac{1}{8}$ chip |
| Number of phases | 2046 | 4092 | 8184 |
| Minimal envelope | 0.75 | 0.87 | 0.93 |
| Maximal SNR loss | 2.5 dB | 1.12 dB | 0.55 dB |

**Table 2.4:** Minimal detected signal envelope (normalized to 1) for different code step sizes.

Figure 2.15 compares the impact of the carrier mismatch $\Delta f$ on the search sensitivity, for two different integration times. The following rule of thumb is generally admitted to fix the frequency step size (also called frequency bin width) as a function of the integration time $T_{int}$ [Kap06]:

$$\Delta f = \frac{2}{3 \cdot T_{int}} \text{ [Hz]} \tag{2.29}$$

The higher the integration time is, the smaller the search frequency bin width must be. The worst frequency mismatch (i.e. $\Delta f = \frac{1}{3 \cdot T_{int}}$) leads to an additional SNR degradation of 1.6 dB.

### Serial search

The classical approach consists in sweeping the two-dimensional search space in a sequential manner (i.e. cell by cell) until a correlation peak is detected. The acquisition time is thus proportional to the number of cells (i.e. the number of code phases times the number of frequency bins). Assuming half a chip code step size and 1 ms coherent integration time ($\Delta f = 667$ Hz), this represents more than 3000 cells to test per satellite.

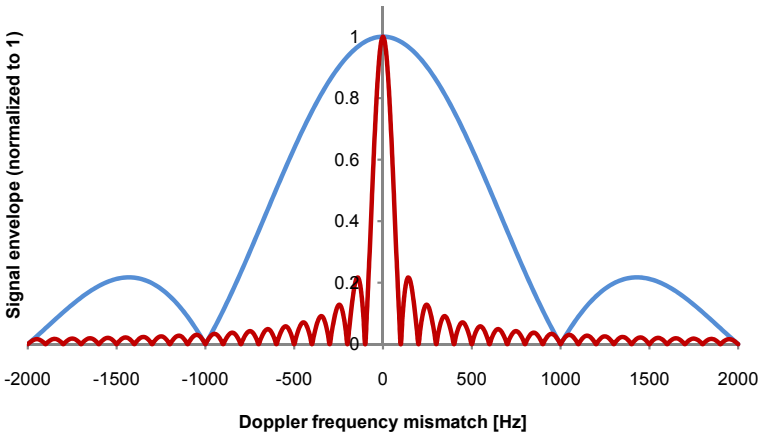


**Figure 2.15:** Signal envelope versus Doppler mismatch for respectively 1 ms (blue curve) and 10 ms (red curve) integration times.

An example of a serial search implementation is illustrated in Figure 2.16.

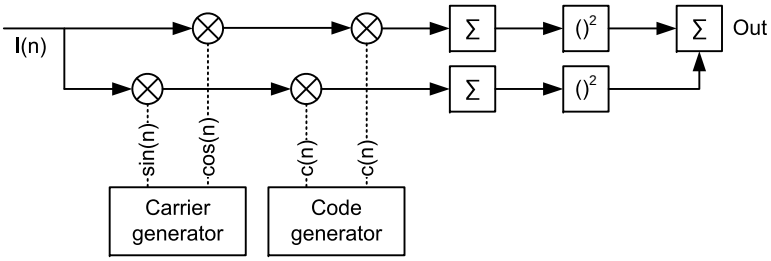


**Figure 2.16:** Serial search architecture. The different code and carrier combinations are sequentially tested.

The serial search is traditionally preferred in hardware receivers, since the high parallelism capabilities of the ASIC can be exploited by implementing many time-shifted code replicas concurrently to test several code phases at once. However, in the case of a software receiver where all the operations are executed sequentially by the microprocessor, performing a serial search is too much time consuming. Therefore, Fast Fourier Transform (FFT) based acquisition methods are preferred for their fastest execution times.

**Parallel code search**

The parallel code search tests all the possible code phases concurrently for a given Doppler frequency. This is done by exploiting the properties of the circular correlation in the frequency domain [Bor07]. The circular correlation of the sequences $x(n)$ and $y(n)$ both with finite length $N$ and with periodic repetition are computed as:

$$z(n) = \frac{1}{N} \cdot \sum_{m=0}^{N-1} x(m) \cdot y(m+n) \tag{2.30}$$

The Discrete Fourier Transform (DFT) of the sequences $x(n)$ and $y(n)$ both with finite length $N$ are computed as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \cdot e^{-j2\pi kn/N} \\ Y(k) &= \sum_{n=0}^{N-1} y(n) \cdot e^{-j2\pi kn/N} \end{aligned} \tag{2.31}$$

The DFT of $z(n)$ can be expressed as:

$$\begin{aligned} Z(k) &= \sum_{n=0}^{N-1} \cdot \sum_{m=0}^{N-1} x(m) \cdot y(m+n) \cdot e^{-j2\pi kn/N} \\ &= \sum_{m=0}^{N-1} x(m) \cdot e^{j2\pi km/N} \cdot \sum_{n=0}^{N-1} y(m+n) \cdot e^{-j2\pi k(m+n)/N} \\ &= X^*(k) \cdot Y(k) \end{aligned} \tag{2.32}$$

where $X^*(k)$ is the complex conjugate of $X(k)$.

The FFT is an algorithm to compute the DFT efficiently. The two abbreviations are used equivalently throughout this document.

The input signal is first multiplied with the carrier to form a complex signal that is transformed into the frequency domain via FFT. The FFT of the locally generated PRN code is also computed. After multiplication of these two sets of coefficients, the inverse FFT is performed to determine if a correlation peak is present. If not, the operation is repeated for the next Doppler frequency bin. An example of a parallel code search implementation is illustrated in Figure 2.17.

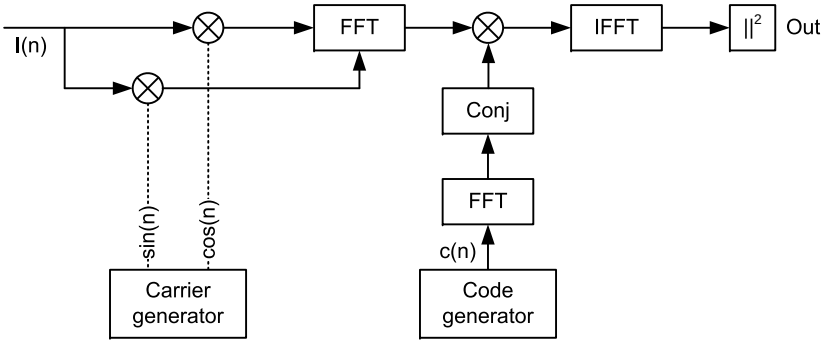


**Figure 2.17:** Parallel code search architecture. For a given Doppler frequency, all the code phases are tested at once.

Compared to the serial search, the parallel code search reduces the two-dimensional space to the different carrier frequencies. As the FFT of the generated PRN codes can be pre-computed and stored off line, each of the frequency bin search consists in performing one FFT and one inverse FFT. However, the frequency shift in the base-band (i.e. multiplication by a complex carrier in the time domain) can be performed in the frequency domain as well, where the multiplication is translated into a convolution with a single Dirac function centered at the estimated Doppler frequency. This corresponds to perform a simple cyclic shift of the FFT frequency components (the spectrum of a FFT is periodic). However, as the Dirac can only be positioned with a limited resolution, a quantization is introduced on the demodulation frequency. The resolution $\delta f$ can be defined as:

$$\delta f = \frac{f_s}{N} \text{ [Hz]} \tag{2.33}$$

The effect of the limited resolution has to be added to the amplitude loss caused by the Doppler frequency mismatch. Consequently, in the worst case, the maximum frequency error is half a Doppler bin, plus half a FFT frequency bin.

The parallel code search method outputs a correlation value for each sample. However, the number of samples per integration period is most likely not a power of two and the input signal must be carefully rearranged with left-zero padding. The advantage of using powers of two for the number of samples is the use of fast algorithms for the FFT computation [Lin06].

Furthermore, due to the unknown code phase of the incoming signal, one additional code period $T_{code}$ is needed to discard the end effect, as shown in Figure 2.18.

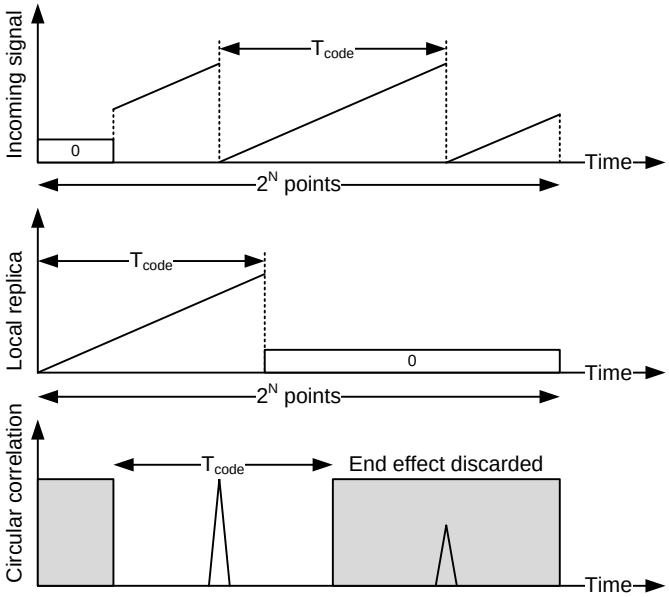


**Figure 2.18:** End effect due to zero padding.

Consequently, the optimal FFT size can be determined as:

$$N = 2^{\lceil \log_2(f_s \cdot (T_{int} + T_{code})) \rceil} \tag{2.34}$$

where $\lceil x \rceil$ denotes the *ceil(x)* function that rounds the value of $x$ towards the nearest largest integer.

That is, for a sampling frequency $f_s = 4.092$ MHz and an integration time $T_{int} = 1$ ms, the FFT size will be $N = 8192$ points, providing a code step size of roughly one fourth of a chip. The sampling frequency must thus be chosen such as to provide a sufficient search phase resolution while keeping the computational complexity to a reasonable level.

A hardware implementation of the parallel code search method would be penalized by prohibitive silicon area requested for implementing large FFTs engine, as well as the relative low speed of the sampling frequency. On the other hand, the software implementation can easily accommodate large FFTs and can take advantage of the Central Processing Unit (CPU) processing power to ensure high speed operations.

### Parallel frequency search

The parallel frequency search tests all the Doppler bins concurrently for a given code phase. The baseband signal is multiplied with the local code replica and accumulated in order to form $N_p$ consecutive pre-detection sums, with a pre-detection time $T_p$ that is $N_p$ times smaller than the integration time $T_{int}$. The $N_p$ results are then regrouped in a vector on which a $N$-point FFT is computed, where $N \geq N_p$ (zero padding is applied if $N > N_p$). Assuming $N_p$ large enough, all the Doppler bins are searched in parallel for a given phase code. If no correlation peak is detected, the operation is repeated with the next code phase. An example of a parallel frequency search implementation is illustrated in Figure 2.19.

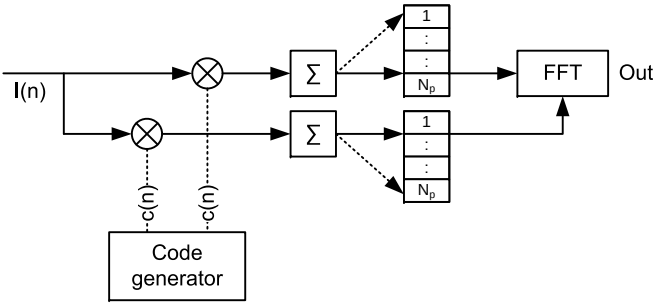


**Figure 2.19:** Parallel frequency search architecture. For a given code phase, all the Doppler frequencies are tested at once.

The system is characterized by the following parameters:

- $N_p \cdot T_p = T_{int}$: total coherent integration time [s];
- $\frac{1}{T_p}$: new sampling rate and search bandwidth of the FFT [Hz];
- $\frac{1}{N_p \cdot T_p}$: frequency bin resolution (or bin width) [Hz].

Each point of the FFT is actually a detector for the maximum of correlation for a given residual frequency. Consequently, the sensitivity of the detection is thus altered by two processes as illustrated in Figure 2.20. The integration time leads to a global power envelope (red curve) proportional to $\text{sinc}^2\ (\pi \cdot \Delta f \cdot T_p)$, while the number of bins, and thus their frequency width, leads to an envelope (blue curve) proportional to $\text{sinc}^2\ (\pi \cdot \Delta f \cdot T_{int})$.

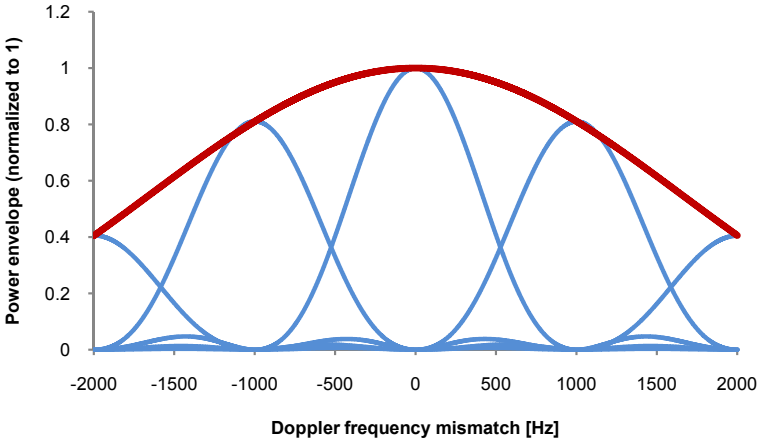


**Figure 2.20:** Sensitivity of a 4-point FFT detector with ± 2 kHz range.

The consequences are:

- the sensitivity reduces when the Doppler mismatch increases. The worst loss occurs when $\Delta f = \frac{1}{2 \cdot T_p}$;
- if the current Doppler shift falls between two Doppler bins, the sensitivity will be even more reduced and two neighboring bins will fire a response. In order to recover this loss, adjacent frequency bins can be combined [Mat03] or [Tsu05].

The FFT should have frequency components covering the ±5 kHz Doppler range in order to find the frequency mismatch in a single pass. Therefore, the signal should be down sampled at a rate of 10 kHz, by using a pre-detection time $T_p = 100\ \mu$s. However, signals with a large Doppler shift will suffer from a strong attenuation. This can be compensated by increasing the coherent integration time $T_{int}$ at the cost of a larger FFT.

Typical FFT configurations are reported in Table 2.5.

| | $N_p$ | $T_p$ | $T_{int}$ | Doppler range | Bin width |
|---|---|---|---|---|---|
| Low sensitivity | 10 (16) | 100 $\mu$s | 1 ms | $\pm$5 kHz | 625 Hz |
| Mean sensitivity | 16 | 100 $\mu$s | 1.6 ms | $\pm$5 kHz | 625 Hz |
| High sensitivity | 128 | 100 $\mu$s | 12.8 ms | $\pm$5 kHz | 78 Hz |

**Table 2.5:** Example of typical parallel frequency search configurations.

### 2.4.2 Tracking algorithms

The acquisition provides a rough estimation of the carrier frequency and the code phase. The tracking algorithms aim to refine these parameters in order to keep track of them over time and so maximize the amplitude of the correlator outputs. If the satellite is properly tracked (i.e. $\Delta\tau = 0$, $\Delta f = 0$, and $\Delta\phi = 0$ in Equation 2.26), both the carrier and the code are removed from the incoming signal and only the data message remains, modulating the sign of $I^x$ over time, as illustrated in Figure 2.21.

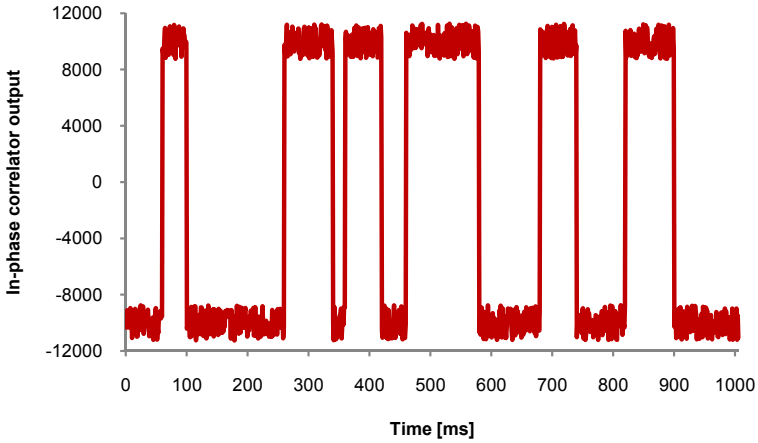


**Figure 2.21:** In-phase correlator output when the satellite is properly tracked. The sign changes represent the data bit transitions.

Many algorithms (hereafter referenced as discriminators) are proposed in the literature for monitoring those parameters (see e.g. [Kap06]) and some are given here as example.

### Carrier frequency discriminator

If the carrier is properly replicated (i.e. $\Delta f = 0$ in Equation 2.26), the correlation results $I^x$ and $Q^x$ contain no frequency component since $\Delta\phi$ is constant. On the other hand, if a carrier mismatch $\Delta f$ subsists, these signals are modulated by a residual frequency and their amplitude affected by a loss. The carrier frequency discriminator $D_f$ estimates the frequency mismatch $\Delta f$ by monitoring the evolution of the vector sum of $I^P$ and $Q^P$ (known as phasor) at times $t_1$ and $t_2$. This is illustrated in the left-hand part of Figure 2.22.

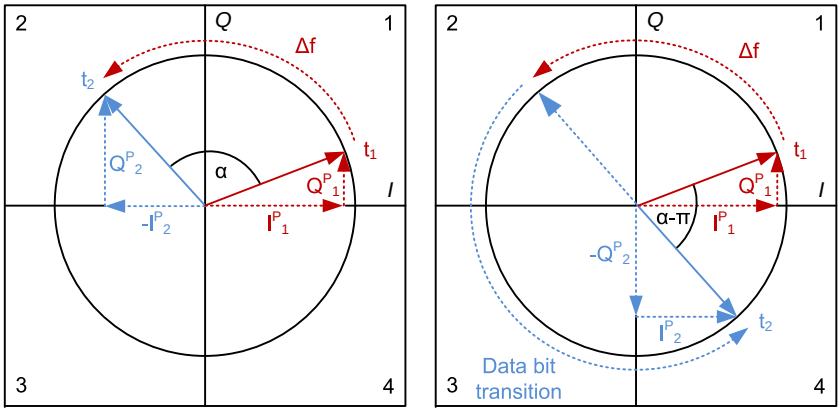


**Figure 2.22:** Phasor diagrams representing the frequency mismatch between the satellite carrier and the local replica (respectively without and with data bit transition).

The phasor rotates proportionally to the frequency error in the tracking loop and the residual frequency is obtained by measuring the phase change $\alpha$ over a fixed time interval $t_2 - t_1$ as:

$$D_f = \frac{\alpha}{2 \cdot \pi \cdot (t_2 - t_1)} = \frac{\arctan 2\ (cross, dot)}{2 \cdot \pi \cdot (t_2 - t_1)}\ [\text{Hz}] \qquad (2.35)$$

where $cross = I_1^P \cdot Q_2^P - I_2^P \cdot Q_1^P = \|\overrightarrow{IQ}_1^P\| \cdot \|\overrightarrow{IQ}_2^P\| \cdot \sin(\alpha)$;
$dot = I_1^P \cdot I_2^P + Q_1^P \cdot Q_2^P = \|\overrightarrow{IQ}_1^P\| \cdot \|\overrightarrow{IQ}_2^P\| \cdot \cos(\alpha)$.

There is no frequency ambiguity as long as the consecutive $I^P$ and $Q^P$ samples are taken within the same data bit interval. In this case, the discriminator can distinguish any phase change $\alpha \in [-\pi;\ \pi]$, defining the range of detectable frequency mismatches to:

$$\frac{-1}{2 \cdot (t_2 - t_1)} < \Delta f < \frac{1}{2 \cdot (t_2 - t_1)} \text{ [Hz]} \tag{2.36}$$

However, if a data bit transition occurs in between $t_1$ and $t_2$, the sign of $I_2^P$ and $Q_2^P$ is inverted with respect to $I_1^P$ and $Q_1^P$, thus introducing an additional phase shift of $\pi$ in the phasor diagram. Consequently, a positive mismatch $\Delta f$ (respectively negative) inducing an absolute phase shift $|\alpha| > \frac{\pi}{2}$ and affected by a data bit transition is no more discernible from a negative mismatch (respectively positive). This phenomenon is illustrated in the right-hand part of Figure 2.22.

### Carrier phase discriminator

If the carrier is properly replicated (i.e. $\Delta f = 0$ in Equation 2.26), the phasor stops rotating. However it may stop at any angle with respect to the I axis, randomly distributing the energy between the in-phase and quadrature signal components, as illustrated in Figure 2.23.

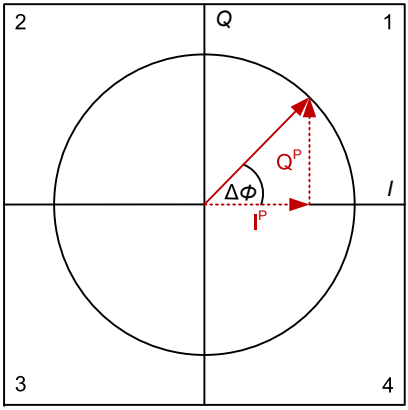


**Figure 2.23:** Phasor diagram representing the phase mismatch between the satellite carrier and the local replica.

The carrier phase discriminator $D_\phi$ estimates this residual phase $\Delta\phi$ by measuring the ratio between $Q^P$ and $I^P$:

$$D_\phi = \arctan\left(\frac{Q^P}{I^P}\right) = \arctan\left(\frac{\sin(\Delta\phi)}{\cos(\Delta\phi)}\right) \text{ [rad]} \tag{2.37}$$

If the phase is properly locked (i.e. $\Delta\phi = 0$), the in-phase component of the signal is maximum (signal plus noise) while the quadrature part tends to zero (noise only). The discriminator is insensitive to the presence of the data modulation and is usually referenced as Costas discriminator, due to the name of its original inventor.

### Code phase discriminator

If the local replica is correctly aligned with the incoming signal code (i.e. $\Delta\tau = 0$ in Equation 2.26), the amplitude of the P correlator output is maximal and those of the E and L components are equal. On the other hand, if a phase mismatch subsists, the two latter outputs are unequal by an amount which is proportional to the phase mismatch, as illustrated in Figure 2.24.

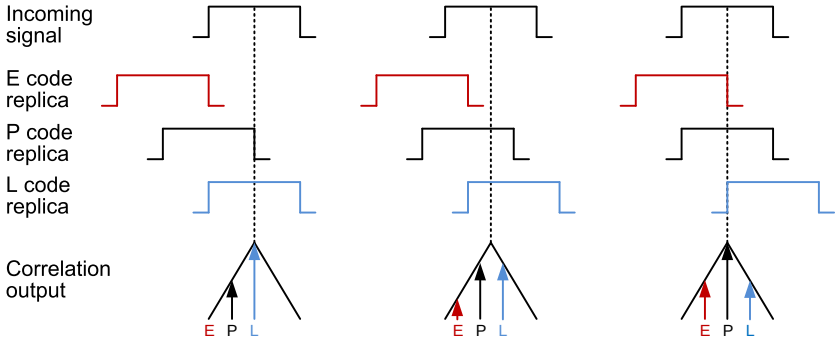


**Figure 2.24:** E, P, and L correlator outputs with respect to the alignment between the incoming signal and the local code replicas [Kap06].

The code phase discriminator $D_\tau$ estimates the code mismatch $\Delta\tau$ by monitoring the amplitude distribution between the E and P correlator outputs. The relation between the correlator outputs and the code mismatch $\Delta\tau$ is illustrated in Figure 2.25.

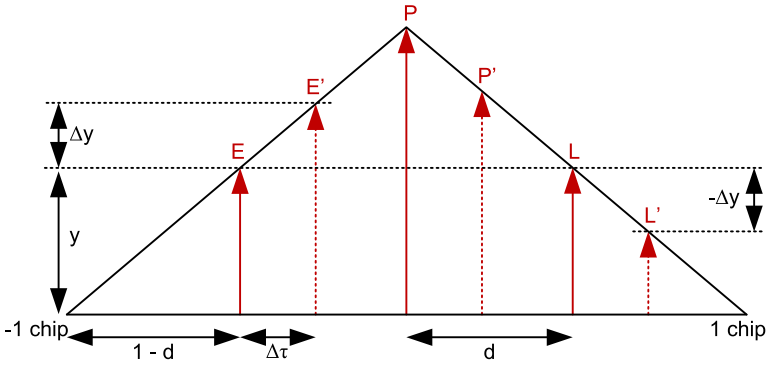


**Figure 2.25:** E, P, and L correlator outputs when the code replica is aligned with the incoming signal (plain arrows), respectively not aligned (dashed arrows).

Thanks to the intercept theorem (also known as Thales' theorem), stating that all corresponding sides of similar triangles are proportional [Sti05], $\Delta\tau$ can be expressed as:

$$\Delta\tau = \frac{(1-d)\cdot\Delta y}{y} \text{ [chips]} \tag{2.38}$$

Assuming a constant chip spacing $d$ between the E and P, and respectively the P and L code replicas, the values $y$ and $\Delta y$ can be expressed as:

$$\begin{aligned} y &= \frac{E'+L'}{2} \\ \Delta y &= \frac{E'-L'}{2} \end{aligned} \tag{2.39}$$

where $E' = \sqrt{(I^E)^2+(Q^E)^2}$;
$L' = \sqrt{(I^L)^2+(Q^L)^2}$.

Both the in-phase and quadrature components are used, making the result independent of the carrier phase (cf. Equation 2.28). The code discriminator $D_\tau$, known as non-coherent early-minus-late envelope, takes the following form:

$$D_\tau = (1-d)\cdot\frac{\sqrt{(I^E)^2+(Q^E)^2}-\sqrt{(I^L)^2+(Q^L)^2}}{\sqrt{(I^E)^2+(Q^E)^2}+\sqrt{(I^L)^2+(Q^L)^2}} \text{ [chips]} \tag{2.40}$$

### Tracking loops

The different discriminator outputs are first filtered in order to reduce the noise and produce a more accurate estimation of the measurements. The filter outputs are then fed back to the base-band processing to update the phase increment of the carrier and code NCOs, in order to reproduce a refined version of the local signals replicas. This whole processing chain constitutes the tracking loop, defined as Frequency Lock Loop (FLL), Phase Lock Loop (PLL), or Delay Lock Loop (DLL) with respect to the type of discriminator used. Generic carrier and code loops are illustrated in Figure 2.26.

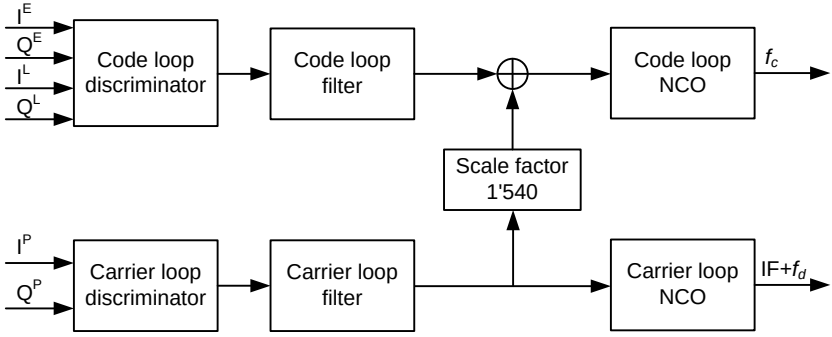


**Figure 2.26:** Typical carrier and code loops, each composed of a discriminator, a filter, and a NCO.

The carrier frequency and phase discriminators can either be integrated into two separate FLL and PLL loops or can be operated continuously and their output combined into a single FLL-assisted PLL loop [War98]. In this case, when the FLL error is zeroed, the filter behaves as a pure PLL and vice versa. For low dynamic environments, the signal lock is done in PLL mode only with a minimal assistance of the FLL, while during acceleration period, the FLL assistance increases and becomes pre-dominant in case of high dynamic environment. An example of a second order lowpass FLL-assisted PLL filter is shown in Figure 2.27.

The carrier filter output is scaled down by a factor 1'540 (corresponding to the ratio between the carrier and code frequencies) and added to the code filter output in order to provide Doppler aiding to the code loop. The carrier loop jitter is less noisy than in the code loop, and the aiding virtually removes all the line of sight dynamics from the code loop [Kap06].

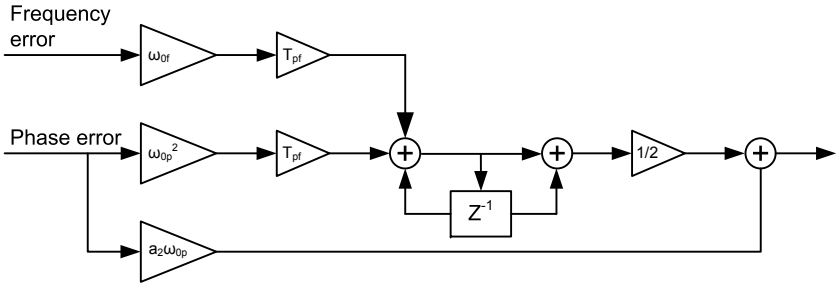


**Figure 2.27:** Block diagram of a second order PLL digital filter assisted by a first order FLL. Since the frequency discriminator output is in unit of Hertz and the phase discriminator in unit of phase, the frequency insertion point into the filter is one integrator back from the phase one [War98].

The code discriminator is integrated in a dedicated DLL loop. An example of a typical second order lowpass filter is shown in Figure 2.28.

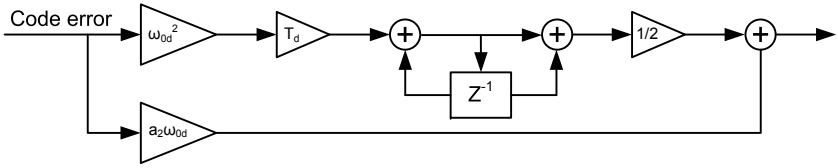


**Figure 2.28:** Block diagram of a second order digital filter, excluding the NCO integrator [Kap06].

## 2.5 Receiver PVT solution computation

The final task of the receiver is to compute the PVT solution, approximately once every second.

### 2.5.1 Principle of the satellite positioning

The user position is determined by trilateration, by computing the distances separating the receiver from the different satellites with known coordinates in space. The exact space vehicle position, obtained from the ephemeris data transmitted in the navigation message, and the distance from the receiver define a spherical surface centered on the satellite. With three satellites defining three spheres, the surfaces intersect at two points, one being the position of the receiver. The principle for two satellites in 2D is illustrated in Figure 2.29.

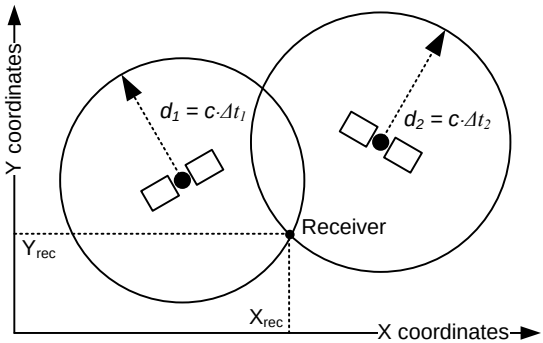


**Figure 2.29:** The position of the receiver is given by the intersection of the two circles of a radius $d_i$ around the satellites $i$ [Zog09].

The distance information $d$ is established by measuring the propagation time $\Delta t$ required by each broadcasted satellite signal to reach the receiver antenna:

$$d = c \cdot \Delta t \text{ [m]} \tag{2.41}$$

where $c \approx 3 \cdot 10^8$ m/s is the speed of light.

By comparing the arrival time of the signal in the receiver with its emission time from the satellite, it is possible to determine $\Delta t$.

Ideally, if both the satellite and the receiver clocks were perfectly synchronized within the same reference time, the exact propagation time would be obtained by subtracting the signal emission time from the satellite $T_{sat}$ from the signal reception time in the receiver $T_{rec}$:

$$\Delta t = T_{rec} - T_{sat} \text{ [s]} \tag{2.42}$$

However, practically this is generally not the case and both the satellite and the receiver clocks have their own bias error from the reference time, respectively $\delta t_{sat}$ and $\delta t_{rec}$, which affects the measurement. The timing relationships are illustrated in Figure 2.30.

where $T_{sat}$: reference time at which the signal leaves the satellite [s];
$T_{rec}$: reference time at which the signal reaches the receiver [s];
$t_{sat} = T_{sat} + \delta t_{sat}$: biased satellite clock reading the time at which the signal leaves the satellite [s];
$t_{rec} = T_{rec} + \delta t_{rec}$: biased receiver clock reading the time at which the signal reaches the receiver [s].

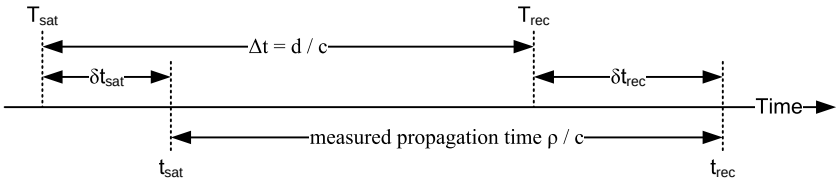


**Figure 2.30:** Timing relationships between the measurements [Kap06].

Thus, the effective range measured $\rho$ is denoted as pseudorange, as it is determined by the time difference between two non-synchronized clocks (satellite and receiver) multiplied by the speed of light:

$$\begin{aligned} \rho &= c \cdot (t_{rec} - t_{sat}) \\ &= c \cdot [(T_{rec} + \delta t_{rec}) - (T_{sat} + \delta t_{sat})] \\ &= d + c \cdot (\delta t_{rec} - \delta t_{sat}) \text{ [m]} \end{aligned} \tag{2.43}$$

The different satellites clock biases are permanently monitored by the GPS ground stations and corrections are broadcasted in the navigation message in order for the receiver to re-synchronize the satellites emission times to the reference time. Assuming that the offset $\delta t_{sat}$ can be compensated, the above equation becomes:

$$\rho = d + c \cdot \delta t_{rec} \text{ [m]} \tag{2.44}$$

Consequently, in order to determine the user position in three dimension $(X_{rec},\ Y_{rec},\ Z_{rec})$, the pseudorange measurements of at least four satellites are required - the fourth satellite necessary for the unknown receiver time offset $\delta t_{rec}$ - resulting in the following system of equations:

$$\begin{aligned}
\rho_1 &= \sqrt{(X_1 - X_{rec})^2 + (Y_1 - Y_{rec})^2 + (Z_1 - Z_{rec})^2} + c \cdot \delta t_{rec} \text{ [m]} \\
\rho_2 &= \sqrt{(X_2 - X_{rec})^2 + (Y_2 - Y_{rec})^2 + (Z_2 - Z_{rec})^2} + c \cdot \delta t_{rec} \text{ [m]} \\
\rho_3 &= \sqrt{(X_3 - X_{rec})^2 + (Y_3 - Y_{rec})^2 + (Z_3 - Z_{rec})^2} + c \cdot \delta t_{rec} \text{ [m]} \\
\rho_4 &= \sqrt{(X_4 - X_{rec})^2 + (Y_4 - Y_{rec})^2 + (Z_4 - Z_{rec})^2} + c \cdot \delta t_{rec} \text{ [m]}
\end{aligned} \quad (2.45)$$

where $(X_i,\ Y_i,\ Z_i)$ are the coordinates of the satellite $i$.

This set of non-linear equations is generally solved for the unknowns by employing iterative techniques based on linearization (see [Kap06]).

### 2.5.2 Pseudoranges measurement

There is a one-to-one relationship between every code chip of the broadcasted signal and the GPS time. The precise time of emission from the satellite is equivalent to the phase of the PRN code with respect to the beginning of the GPS week. When the broadcasted signal reaches the receiver, which is accurately reproducing the code replica correlating with it, the phase offset of the replicated code with respect to the beginning of the GPS week represents the emission time $t_{sat}$ of the satellite signal. This is illustrated in Figure 2.31.

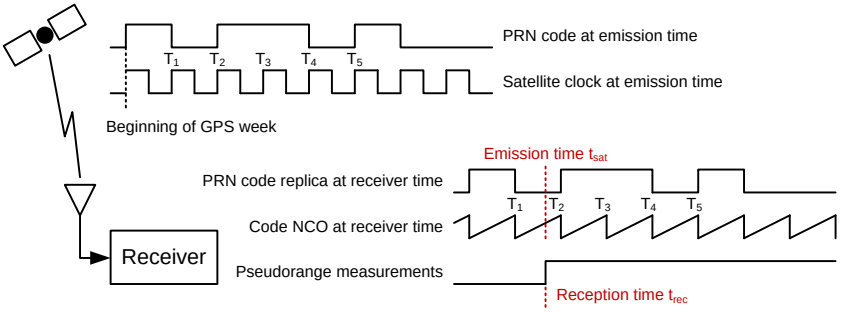


**Figure 2.31:** Timing relation between the satellite emission time $t_{sat}$ and the receiver reception time $t_{rec}$ [Kap06].

When the receiver takes a set of pseudoranges measurements at time $t_{rec}$ - with respect to its own internal clock - the base-band processing records the current code NCO accumulator state. This provides the fractional part of the currently generated chip within the code epoch. Since the CA code repeats every millisecond and the time delays from the satellites are in the range of 67 to 86 ms (for a user on the surface of the earth), there is a GPS time ambiguity every millisecond. This is solved by decoding the timing information provided in the navigation message in order to reconstruct $t_{sat}$ with respect to the GPS time, as illustrated in Figure 2.32.

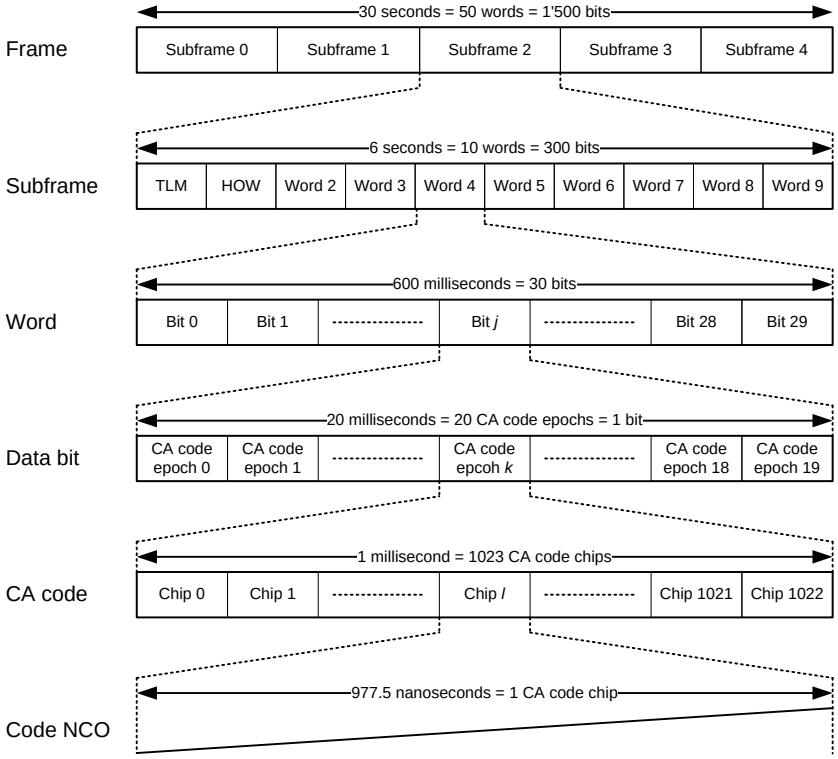


**Figure 2.32:** GPS L1 CA code timing relationships [Kap06].

## 2.6 Summary

Chapter 2 analyzes the structure of the broadcasted GPS L1 CA signal and then reviews the successive receiver operations involved in its demodulation, from the antenna up to the PVT solution computation. The different processing stages are described from a traditional hardware receiver point of view, in order for the reader to fully apprehend the specific constraints inherent to a software implementation introduced in Chapter 3.

# Chapter 3

# Challenges of a software receiver

The purpose of the software receiver is to entirely process the RF front-end digital output stream on a general purpose microprocessor. This includes all the base-band operations traditionally performed by an ASIC, as well as the base-band algorithms and the PVT solution computation. The implementation of the whole data processing chain in software imposes specific constraints as compared to a classical hardware receiver.

## 3.1 Data rate

The ideal software receiver would place the ADC as close as possible to the antenna for digitizing the incoming signal at twice the carrier frequency. In addition to the practical constraints of its implementation, the direct sampling generates a minimal data rate of 3.152 Gb/s (cf. Equation 2.14), which is for now a prohibitive rate for a microprocessor to handle. More generally, the minimal data rate is fixed by the Nyquist criterion and the data quantization. In order to capture the main lobe null-to-null bandwidth of the CA code, the requested sampling frequency is at least 4.092 MHz for real input samples or 2.046 MHz for complex ones [Tsu05]. Assuming a 2-bit incoming signal, this represents a minimal theoretical data throughput of 8.184 Mb/s between the ADC output and the host system. Several common interfaces are available for embedded applications and Table 3.1 provides a non exhaustive list.

| | Data rate [Mb/s] |
|---|---|
| Controller Area Network (CAN) | 1 |
| RS-232 | 1.5 |
| Microwire | 3 |
| Inter-Integrated Circuit ($I^2C$) | 3.4 |
| Serial Peripheral Interface (SPI) | 10 |
| Firewire (IEEE 1394a/b) | 400/800 |
| Universal Serial Bus (USB) | 12 |
| Universal Serial Bus (USB) 2.0 | 480 |

**Table 3.1:** Maximal theoretical data rate of common host interfaces.

The Serial Peripheral Interface (SPI) and the Universal Serial Bus (USB) interfaces can handle the minimal requested data rate of 8.184 Mb/s. However, only the Firewire and the USB 2.0 interfaces can accommodate higher sampling frequencies or data bit-depths.

## 3.2 Computational load

The base-band processing is the heart of the receiver and constitutes the most power-demanding task, since it manipulates the data at the system sampling rate of several megahertz, in parallel for several satellite channels. As a microprocessor handles the data in a sequential manner and has not the ASIC capability of processing them in parallel (unless a multi-core architecture or specific parallel instruction schemes are used), this represents a heavy computational burden. In comparison, the base-band algorithms and the PVT solution computation are executed at a much lower rate (less than one kilohertz) and have thus a quasi negligible impact on the global system performances. In order to fully apprehend the challenges of implementing a GPS receiver in software, let us try to estimate the amount of integer operations per second involved in the base-band processing. Referring to the generic complex architecture depicted in Figure 2.13, the following tasks are executed in parallel for the $N_c$ satellite channels at the system sampling frequency:

1. synthesize the complex carrier at the $IF$ (plus Doppler) frequency;
2. multiply the incoming complex signal with the local complex carrier;
3. synthesize the E, P, and L code replicas;
4. multiply the complex base-band signal with the E, P and L code replicas;
5. accumulate the six resulting signals separately over the integration period to form the $I^E$, $I^P$, $I^L$, $Q^E$, $Q^P$, and $Q^L$ correlation results.

The synthesis of the carrier and code replicas is traditionally achieved by means of a NCO, as described in Subsection 2.3.1. This requires to update a phase accumulator at the system sampling rate, and in parallel for several satellite channels. A former study estimates that the dynamic generation of the digital sinusoid absorbs up to 30% of the time to generate a correlation [Hec04]. Consequently, the implementation of a conventional NCO in software is not adapted and makes the real-time signals generation computationally too expensive.

For the sake of simplicity, we ignore the cost of generating the local carrier and code sequences and focus on the multiply and accumulate operations involved in the mixing and accumulation operations in Equation 2.20 and Equation 2.23. In addition to the unavoidable load and store operations, Table 3.2 provides a rough estimation of the amount of integer operations per second necessary to process $N_c$ satellite channels in the base-band:

| | # additions | # multiplications |
|---|---|---|
| Carrier mixing | $2 \cdot f_s \cdot N_c$ | $4 \cdot f_s \cdot N_c$ |
| Code mixing | - | $6 \cdot f_s \cdot N_c$ |
| Accumulation | $6 \cdot f_s \cdot N_c$ | - |
| Total | $8 \cdot f_s \cdot N_c$ | $10 \cdot f_s \cdot N_c$ |

**Table 3.2:** Amount of integer operations per second involved in a complex base-band architecture, assuming three code replicas(E, P and L).

Assuming a sampling frequency $f_s$=4.092 MHz and a 12 satellite channels receiver configuration, the base-band processing requires approximately $3.9 \cdot 10^8$ raw additions and $4.9 \cdot 10^8$ multiplications to perform each second.

Depending on the processor architecture, the execution time of an integer operation may fluctuate significantly. Table 3.3 provides the number of clock cycles required for the core to execute all the micro-operations that form an instruction (or latency) for different mass market processors:

| | Intel Pentium 4 | Intel Core Duo | Intel Core i7 |
|---|---|---|---|
| Addition | 1 | 1 | 1 |
| Multiplication | 14 | 4 | 3 |

**Table 3.3:** Latency of an integer addition and multiplication expressed in processor clock cycles [Int09].

Depending on the processor, the operations of Table 3.2 require at least $1.9 \cdot 10^9$ clock cycles per second, without including the load of the carrier and code generation. Furthermore, on a computer with a modern multi-tasking operating system, the resources are spread among many different programs running simultaneously. In order to retain the responsiveness of the system, it is therefore advisable to limit the processing load of the GPS software receiver to the half of the total CPU time. Consequently, in order for the receiver to operate in real-time, the processing of one millisecond of data must be achieved within half a millisecond, including the processing of all the present satellites. By restricting the receiver activities to 50% of the processor time, the base-band operations require a system clock faster than 4 GHz. Consequently, a straightforward transposition of conventional hardware-based architectures into software leads to an amount of operations that cannot be handled by today's fastest computers [Hec06]. From this statement, new strategies have to be considered in order to lower the computational load of the base-band operations.

## 3.3 Oscillator drift

Generally, a single crystal oscillator serves as a reference in the receiver, from which the desired frequencies of the different mixers are derived. Most of the crystals oscillate in the range of a tenth or a few hundreds of megahertz and are thus combined with a PLL to achieve the desired higher frequency needed by the Local Oscillator (LO) for the GPS L1 carrier down-conversion. In addition, the reference frequency is often divided to feed the ADC and serves as sampling clock. So any frequency error/drift of the crystal oscillator will affect the receiver at different points.

Table 3.4 provides the characteristics of frequency stability for a Crystal Oscillator (XO) and a Temperature Controlled Crystal Oscillator (TCXO), two types of oscillators commonly found in consumer electronic products.

| | XO | TCXO |
|---|---|---|
| Initial stability | 7 ppm | 1 ppm |
| Temperature stability | 15 ppm | 1 ppm |
| Aging stability | 1 ppm / year | 2 ppm / year |

**Table 3.4:** Typical frequency stability for a XO and a TCXO [Rak10].

Assuming a one year old oscillator, the worst case frequency stability reaches 23 ppm for the XO and 4 ppm for the TCXO.

The oscillator instability first impacts the carrier down-conversion process by modifying the frequency of the local oscillator(s). This translates into an equivalent Doppler shift on the carrier frequency. The effect is maximal in the first RF down-conversion stage, where the local oscillator generates a frequency in the vicinity of the GPS L1 carrier. The equivalent Doppler shift $\Delta f_d$ can be estimated as:

$$\begin{aligned} \Delta f_d &\approx 1.575 \cdot 10^9 \cdot 23 \cdot 10^{-6} = 35 \text{ kHz} && \text{for the XO} \\ \Delta f_d &\approx 1.575 \cdot 10^9 \cdot 4 \cdot 10^{-6} = 6 \text{ kHz} && \text{for the TCXO} \end{aligned} \tag{3.1}$$

Consequently, the IF can be off by up to $\pm 35$ kHz, respectively $\pm 6$ kHz, in addition to the unavoidable $\pm 5$ kHz Doppler shift induced by the satellite motion. This applies for both software and hardware receivers and affects the search range in the acquisition procedure. However, the hardware receivers are generally driven by a dedicated quartz with a better stability as compared to the software ones that are integrated in a low cost system, which may be driven by a clock that is not optimized for ranging applications. Therefore, it is desirable for a software receiver to extend the Doppler frequency uncertainty to $\pm 40$ kHz in order to compensate for any possible offsets.

The clock drift also affects the ADC and the resulting sampling frequency. This is not really an issue for conventional GPS receivers, simply because the whole digital processing chain is driven by the same imperfect frequency. The incoming samples and the locally generated sequences (i.e. the carrier and code replicas) are subject to the same clock bias and the time synchronization between them is kept. However, in a software receiver, the incoming signal is digitized at the sampling clock rate subject to drift, while the local reference signals are software-generated according to the nominal sampling frequency value. Consequently, the time separating two consecutive samples differs between the ADC output stream and the locally generated sequences [Yan03]. This phenomenon is specific to software receivers and translates into an equivalent Doppler shift which mainly affects the code frequency, as illustrated in Figure 3.1.

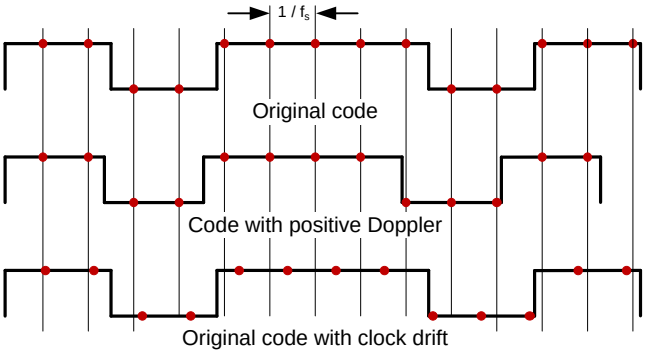


**Figure 3.1:** Effect of the Doppler on the code and equivalent sampling times [Tsu05].

If the sampling frequency is decreased, the time separating two consecutive samples is extended and so produces a data sequence where the code is compressed. A similar way to represent the phenomenon is to keep the nominal sampling frequency and apply a positive Doppler $\Delta f_c$ that shrinks the code in time. Both the sequences result in the same digital values. The equivalent Doppler $\Delta f_c$ on the code induced by a sampling frequency drift can be estimated as (see [Tsu05]):

$$\Delta f_c = -f_c \cdot \frac{\Delta f_s}{\Delta f_s + f_s} \text{ [Hz]} \tag{3.2}$$

where $\Delta f_s$ is the sampling frequency variation [Hz].

If the sampling frequency increases ($\Delta f_s > 0$), then the equivalent Doppler decreases ($\Delta f_c < 0$) and vice-versa. The quartz instability thus induces a code frequency drift $\Delta f_c = 23$ Hz, respectively $\Delta f_c = 4$ Hz, in addition to the unavoidable $\pm 3.2$ Hz Doppler shift due to the satellite motion. In the worst case, this may lead to a large code phase shift of one chip every 40 ms that needs to be compensated either by the local generation process or by adapting the algorithms of the tracking program [Sch02].

## 3.4 Summary

Chapter 3 identifies the main constraints inherent to the software implementation of a GPS receiver. As compared to the traditional hardware solution, specific problems such as the data throughput or the oscillator drift have to be tackled. However, the main challenge resides in the computational burden of the base-band operations, since a microprocessor has not the ASIC capability of manipulating multiple data in parallel. Consequently, new strategies have to be developed in order to lower the complexity of the receiver and make its real-time functioning possible. The main solutions currently proposed in the literature are reviewed in Chapter 4.

# Chapter 4

# Existing architectures of a software receiver

The tasks carried out by an ASIC are too computationally burdensome to be implemented directly on a microprocessor and in that sense, new strategies have to be developed in order to minimize the complexity of the operations involved in the base-band processing. This chapter reviews the main solutions proposed in the literature.

## 4.1 Alternate data processing

Several alternatives to the integer arithmetic of Table 3.2 have been proposed in the literature to lower the computational burden of the base-band processing, [Bar09], and [Bar10]. Some rely on the code optimization using specific CPU functionalities for improving the efficiency and speed of the data processing, while the others exploit the native bitwise representation of the signal and use logical instructions. The different approaches are detailed in the next sections.

### 4.1.1 Single Instruction Multiple Data

The Single Instruction Multiple Data (SIMD) are specific arithmetic CPU instructions that operate on vectors of data. Unlike the standard instructions, the data are manipulated in blocks and several values can be loaded and processed simultaneously to achieve data level parallelism.

In other words, if the SIMD system works by loading, for example, four data values at once, the operation being executed will apply to all the four values at the same time, as illustrated in Figure 4.1.

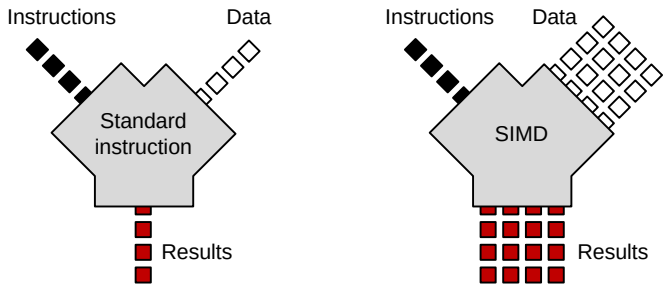


**Figure 4.1:** Standard instruction versus SIMD [Cha07].

*Intel* deployed the first instance of SIMD under the name of *MMX* in 1995 with their *Pentium* line of microprocessors. The main usage of the *MMX* instructions set relies on packed data, which means that instead of using the whole register for a single 64-bit integer, two 32-bit integers, four 16-bit integers, or eight 8-bit integers may be processed concurrently [Int97]. Further extensions integrating floating point arithmetic were introduced later to the $x86$ architecture under the name of Streaming SIMD Extension (SSE) and consist of SSE1 to SSE5. On average, the SIMD operations require more clock cycles to execute than the traditional $x86$ instructions. However, since they operate on multiple values at the same time, SIMD can result in significant gains in execution speed, especially for repetitive and parallel tasks like the ones involved in the base-band processing. By optimizing the receiver code by means of SIMD, [Hec04] claims more than 600% performance improvement with respect to the standard integer implementation. Many software receivers proposed in the literature rely on the use of SIMD, such as for example [Cha06] or [Pan03].

However, the instruction sets are CPU architecture specific and old processors or non-$x86$ processors lack SSE entirely. Similarly, the next generation instructions sets from *Intel* and *AMD* (respectively *AVX* and SSE5) will most likely be incompatible with each other, forcing the programmers to develop different software versions. Consequently, SIMD operations are tied to specific implementations, which severely limit the portability of the code.

### 4.1.2 Instruction pipelining

The basic idea of the instruction pipelining is to split the processing of a CPU instruction into a sequence of micro-operations that can be working on the different stages of several instructions simultaneously. This way, the processor does not wait until all the micro-operations are completed to introduce the next instruction into the pipeline. The principle is illustrated in Figure 4.2.

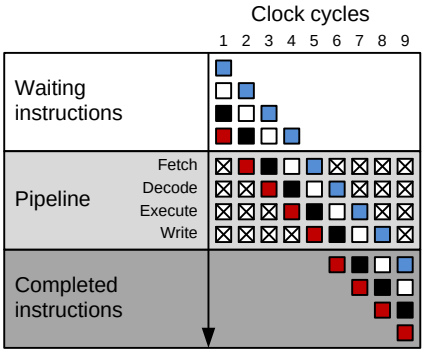


**Figure 4.2:** Generic four stages pipeline.

Consequently, pipelining does not reduce the time needed to complete an instruction (or instruction latency), but increases the number of instructions that can be executed per unit of time (or instruction throughput). Since in many cases, the instruction throughput can be significantly less than its latency, it increases the amount of data that can be processed at once. Table 4.1 regroups the throughput of the addition and the multiplication instructions for different mass market processors.

| | Intel Pentium 4 | Intel Core Duo | Intel Core i7 |
|---|---|---|---|
| Addition | 0.5 | 0.5 | 0.33 |
| Multiplication | 3 | 1 | 0.5 |

**Table 4.1:** Throughput of an integer addition and multiplication expressed in CPU clock cycles. The throughput given in fraction of a clock cycle occurs for multi-speed arithmetic units [Int09].

Let us take the example of two consecutive multiplications performed on an *Intel Pentium 4*. With a latency of 14 clock cycles per multiplication (cf. Table 3.3), the two operations normally take 28 clock cycles to execute. However, the processor can take advantage of the low throughput of the multiplication and schedule the second operation three clock cycles later. Thanks to pipelining, multiplications will therefore only take $14 + 3 = 17$ clock cycles to operate, assuming that the operands of the second instruction do not depend on the result from the first operation.

The performance of a pipelined processor is hard to predict, since the scheduling of the instructions relies on the dependency between the current operand and the results of the previous operation. In order to take full advantage of the pipelining, the structure of the program has to be adapted and the operations have to be carefully reordered [Hec06].

### 4.1.3 Digital Signal Processor

A separate class of processors exists for achieving data level parallelism, commonly referred to as Digital Signal Processor (DSP). While SIMD extensions rely on the general-purpose portions of the CPU to handle the program details, the DSPs are self-contained processors with their own instructions set. They offer great code optimization possibilities, as some are capable of performing several multi-bit multiplications in parallel. Generally reserved to video or sound processing, DSPs are also well suited for executing the correlation operations involved in a software receiver. For example [Hum06] developed a receiver based on a 720 MHz *Texas Instrument* DSP capable of running 43 satellite channels in parallel. Other examples can be found in [Tia08] or [Ako01b].

With the perspective of developing a platform-independent software receiver, the solutions based on the code optimization or use of specific CPU architectures will not be further considered in this document.

### 4.1.4 Bitwise processing

Contrary to SIMD operations, bitwise processing (sometimes also referred in the literature as vector processing) uses universal CPU instructions and exploits the native bit representation of the signal. The different data bits are stored in separate words - generally one sign and one or several magnitude words - between which logical operations are performed independently.

Thus, while the integer arithmetic interprets the data horizontally as a single value, the bitwise processing manipulates the bits separately in a vertical way, as illustrated in Figure 4.3.

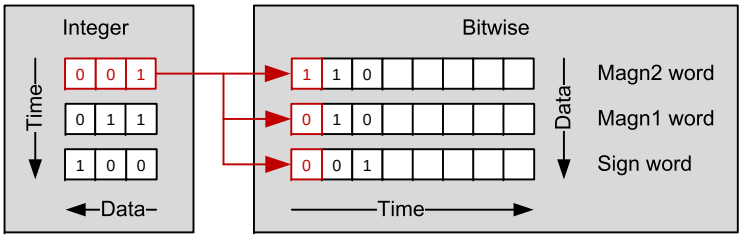


**Figure 4.3:** 3-bit integer versus bitwise data representation.

The objective is to take advantage of the high parallelism and speed of the bitwise operations for which a single integer operation translates into one or a few simple logical relations performed in parallel on all bits of the word. For example, the C programming language supports bitwise manipulation of words by means of the operators listed in Table 4.2.

| Symbol | Operation |
|---|---|
| & | logical AND |
| \| | logical OR |
| ∧ | logical XOR |
| ∼ | logical complement |
| ≫ | logical right shift |
| ≪ | logical left shift |

**Table 4.2:** Available logical operators in C programming language.

Depending on the receiver configuration, the code and carrier mixing can be carried out by a few basic logical operations, performed in parallel on several samples, thus making the bitwise processing particularly efficient. To illustrate this, let us consider the example of a real base-band architecture, as shown in Figure 2.13.

We assume here that the incoming signal $I(n)$ is digitized with 2 bits per sample $\{I_1(n),\ I_0(n)\}$, associated to the integer values $\{\pm 1,\ \pm 3\}$. In the receiver, $I(n)$ is first mixed with a complex carrier $K(n)$ quantized with 2 bits $\{K_1(n),\ K_0(n)\}$, associated to the integer values $\{\pm 1,\ \pm 2\}$. The mixing results in the base-band complex signal $I_{bb}(n)$ and $Q_{bb}(n)$, taking one of the integer values of Table 4.3 [Led04]. For notation simplicity, we only provide the equations for the in-phase components (the same operations apply for the quadrature components).

| | | Local carrier $K(n)$ | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | -1 | -2 |
| Signal $I(n)$ | 1 | 1 | 2 | -1 | -2 |
| | 3 | 3 | 6 | -3 | -6 |
| | -1 | -1 | -2 | 1 | 2 |
| | -3 | -3 | -6 | 3 | 6 |

**Table 4.3:** Integer output values $I_{bb}(n)$ of the carrier mixer.

The table output is encoded with 3 bits $\{I_{bb,2}(n),\ I_{bb,1}(n),\ I_{bb,0}(n)\}$ as shown in Table 4.4.

| | | Local carrier $K_{1,0}(n)$ | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 10 | 11 |
| Signal $I_{1,0}(n)$ | 00 | 000 | 001 | 100 | 101 |
| | 01 | 010 | 011 | 110 | 111 |
| | 10 | 100 | 101 | 000 | 001 |
| | 11 | 110 | 111 | 010 | 011 |

**Table 4.4:** Binary output code $\{I_{bb,2}(n),\ I_{bb,1}(n),\ I_{bb,0}(n)\}$ of the carrier mixer.

The truth table of Table 4.4 translates into the following logical operations between the different sign and magnitude bits:

$$\begin{aligned} I_{bb,2}(n) &= I_1(n) \oplus K_1(n) \\ I_{bb,1}(n) &= I_0(n) \\ I_{bb,0}(n) &= K_0(n) \end{aligned} \tag{4.1}$$

The carrier down-conversion is reduced to two logical operations which can be performed concurrently on all the bits of the words. The PRN code removal simply consists in a sign inversion, respectively non-inversion, which translates into an exclusive OR between the code $c^x(n)$ and the sign words:

$$I_2^x(n) = I_{bb,2}(n) \oplus c^x(n) \tag{4.2}$$

where $x \in \{E,\ P,\ L\}$.

Consequently, the complete carrier and code removal requires only eight operations, for both in-phase and quadrature components and for all the code replicas.

The next step consists in accumulating the different mixing results over the integration period in order to form the correlation values. This is done by counting the number of samples $I^x(n)$ equal to $\pm 1$, $\pm 2$, $\pm 3$, and $\pm 6$. However, as this information is vertically spread over the different sign and magnitude words, a reconversion into the integer representation is finally needed to perform the accumulation and the data readout. Thus, for each of the eight above possible values, a new word is computed and contains a '1' when the current operand equals the tested value and a '0' otherwise. This is done with the following logic relations [Led06b]:

$$\begin{aligned} P1^x(n) &= I_2^x(n) \cdot \overline{I_1^x(n)} \cdot \overline{I_0^x(n)} \\ M1^x(n) &= \overline{I_2^x(n)} \cdot \overline{I_1^x(n)} \cdot \overline{I_0^x(n)} \\ P2^x(n) &= I_2^x(n) \cdot \overline{I_1^x(n)} \cdot I_0^x(n) \\ M2^x(n) &= \overline{I_2^x(n)} \cdot \overline{I_1^x(n)} \cdot I_0^x(n) \\ P3^x(n) &= I_2^x(n) \cdot I_1^x(n) \cdot \overline{I_0^x(n)} \\ M3^x(n) &= \overline{I_2^x(n)} \cdot I_1^x(n) \cdot \overline{I_0^x(n)} \\ P6^x(n) &= I_2^x(n) \cdot I_1^x(n) \cdot I_0^x(n) \\ M6^x(n) &= \overline{I_2^x(n)} \cdot I_1^x(n) \cdot I_0^x(n) \end{aligned} \tag{4.3}$$

By taking advantage of the redundancy and storing intermediate results, Equation 4.3 can be carried out with 15 operations. The next step is to count the number of '1' contained in each of the so created eight words. Many algorithms, such as [Bee72], or even specific CPU instructions exist to perform this operation. However, a very straightforward solution consists in implementing a Look Up Table (LUT) that is directly addressed by the word itself and that outputs the number of '1' contained in the address. The principle is illustrated in Figure 4.4.

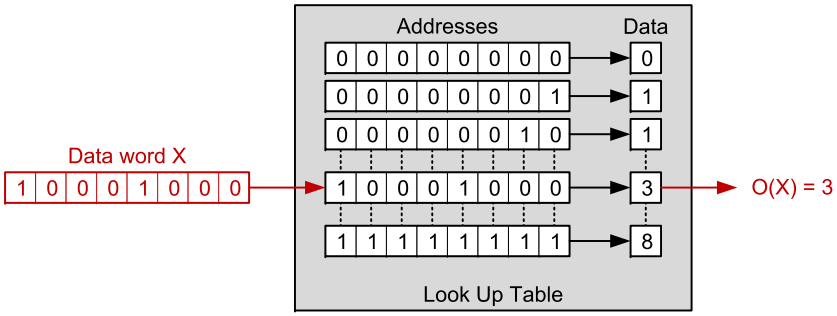


**Figure 4.4:** LUT outputting the number of '1' contained in the 8-bit input address.

The table must be able to fit into the microprocessor cache to allow fast execution, but must be also large enough to minimize the memory accesses; 64 kB (16-bit addressing) is typically a good compromise.

Finally, the correlation results can be expressed as the following linear combination:

$$\begin{aligned} I^x = {} & 6 \cdot [O(P6^x) - O(M6^x)] + 3 \cdot [O(P3^x) - O(M3^x)] \\ & + 2 \cdot [O(P2^x) - O(M2^x)] + [O(P1^x) - O(M1^x)] \end{aligned} \tag{4.4}$$

where $O()$ is the operator counting the number of '1'.

In addition to unavoidable load and store operations, Table 4.5 provides a rough estimation of the amount of operations per second necessary to process $N_c$ satellite channels in the base-band (without carrier and code generation). Note that the efficiency of the bitwise processing depends on the register size of the host computer (8, 16, 32, 64 or 128 bits) which fixes the maximal number of bits $N_b$ processed per operation.

| | # bitwise operations | # LUT accesses | # additions |
|---|---|---|---|
| Carrier mixing | $2 \cdot f_s \cdot N_c/N_b$ | - | - |
| Code mixing | $6 \cdot f_s \cdot N_c/N_b$ | - | - |
| Accumulation | $90 \cdot f_s \cdot N_c/N_b$ | $48 \cdot f_s \cdot N_c/N_b$ | $48 \cdot f_s \cdot N_c/N_b$ |
| Total | $98 \cdot f_s \cdot N_c/N_b$ | $48 \cdot f_s \cdot N_c/N_b$ | $48 \cdot f_s \cdot N_c/N_b$ |

**Table 4.5:** Amount of operations per second involved in the carrier mixing, the code mixing and the accumulation.

As compared to a traditional implementation such as described in Table 3.2, the integer multiplication, which requires several clock cycles in a standard microprocessor, is advantageously replaced by logical operations that execute rapidly and in parallel on the $N_b$ word bits. This makes the bitwise processing particularly efficient and well fitted for a software implementation. However, the complexity of the receiver becomes now bit-depth dependent and increases drastically with the number of bits used for the signal and the carrier quantization (any additional bit doubles the size of Table 4.3 and consequently the number of words to compute in Equation 4.3). Furthermore, the execution speed is related to the processor register width, which fixes the number of bits $N_b$ operated in parallel. In conclusion, the inherent drawback of the bitwise processing is the lack of flexibility and scalability as compared to integer operations.

### 4.1.5 Distributed arithmetic

The original concept of distributed arithmetic was developed for optimizing the implementation of digital filters on a FPGA [Act10]. The main idea is to rearrange the multiplies and adds of a sum of products at the bit level to take advantage of small tables of pre-computed sums [And99]. However, we propose here to extend the concept to a GNSS software receiver design in order to optimize the accumulations involved in the correlation process, [Wae09a], and [Wae10].

The accumulation consists in summing up the consecutive samples of the signals $I^E(n)$, $I^P(n)$, and $I^L(n)$ and $Q^E(n)$, $Q^P(n)$, and $Q^L(n)$ after the code removal, as written in Equation 2.23:

$$I^x = \sum_{n=0}^{N_s-1} I_{bb}(n) \cdot c^x(n) = \sum_{n=0}^{N_s-1} I^x(n) \tag{4.5}$$

where $x \in \{E,\ P,\ L\}$;
$N_s$ is the number of samples per integration.

For notation simplicity, we only provide the equations for the in-phase components (the same operations apply for the quadrature components). Let us express the signal $I^x(n)$ as a linear combination of its $M$ quantization bits. We consider here the two's complement notation which decomposes the signal as follows:

$$I^x(n) = -2^{M-1} \cdot I^x_{M-1}(n) + \sum_{m=0}^{M-2} 2^m \cdot I^x_m(n) \tag{4.6}$$

where $I^x_m(n)$ is the $m^{th}$ bit of the signal $I^x(n)$ at sampling time $n$.

We define the sum $S^x_m$ associated to the $m^{th}$ data bit $I^x_m(n)$ of the signal $I^x(n)$ as:

$$S^x_m = \sum_{n=0}^{N_s-1} I^x_m(n) \tag{4.7}$$

where $m \in [0,\ M-1]$.

By combining and rearranging the terms of the above three equations, we finally obtain:

$$\begin{aligned} I^x &= \sum_{n=0}^{N_s-1} \left( -2^{M-1} \cdot I^x_{M-1}(n) + \sum_{m=0}^{M-2} 2^m \cdot I^x_m(n) \right) \\ &= -2^{M-1} \cdot S^x_{M-1} + \sum_{m=0}^{M-2} 2^m \cdot S^x_m \end{aligned} \tag{4.8}$$

The accumulation $I^x$ is now expressed as a linear combination of $M$ sums $S^x_m$. The challenge now consists in efficiently computing Equation 4.7 for the $M$ bits of $I^x(n)$. Since $S^x_m$ is the arithmetic sum of all the $m^{th}$ bits $I^x_m(n)$ over one integration period, it can be estimated by counting the number of bits equal to the logical value '1'. One straightforward solution is to implement a LUT addressed by $I^x_m(n)$, as illustrated in Figure 4.4.

An example of distributed arithmetic architecture is provided in Figure 4.5.

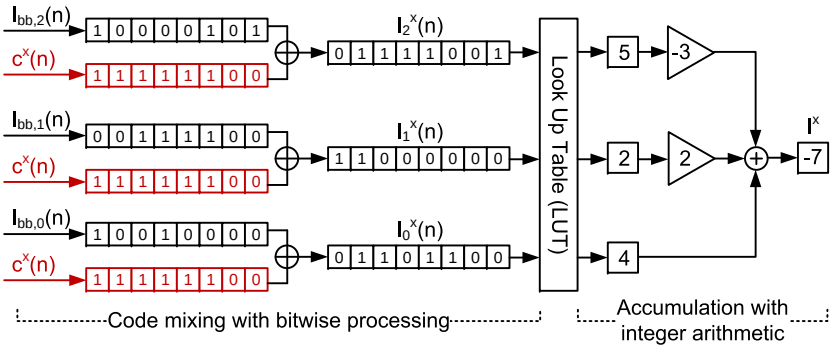


**Figure 4.5:** Example of distributed arithmetic with a 3-bit base-band signal $I_{bb}(n)$.

Thanks to the properties of the distributed arithmetic, the conversion from the bitwise representation into the integer one is automatically performed in parallel to the accumulation. No additional operations are required. Furthermore, the architecture can easily accommodate various signal configurations as the complexity stays almost proportional to the bit-depth of the incoming signal $I(n)$. However, the distributed arithmetic requires the signals to accumulate $I^x(n)$ to be expressed as a linear combination of its $M$ data bits. This introduces an additional constraint on the antecedent bitwise processing, which may necessitate more complex logical operations and more bits to represent $I^x(n)$. To illustrate this point, let us come back to the example of a 2-bit incoming signal mixed with a 2-bit complex carrier, as described in Subsection 4.1.4. The possible integer output values of the carrier mixer are summarized in Table 4.3.

The appropriate binary representations of $I^x(n)$ must be chosen in order to minimize the antecedent number of bitwise operations necessary to perform the carrier and code mixing. In this case, 4 bits are needed to represent the content of Table 4.3 with respect to the following (heuristic) bits decomposition:

$$I_{bb}(n) = -6 \cdot I_{bb,3}(n) + 3 \cdot I_{bb,2}(n) + 2 \cdot I_{bb,1}(n) + I_{bb,0}(n) \tag{4.9}$$

With the above representation, Table 4.3 is encoded with 4 bits, as shown in Table 4.6.

| | | Local carrier $K_{1,0}$(n) | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 10 | 11 |
| Signal $I_{1,0}$(n) | 00 | 0001 | 0010 | 1110 | 1101 |
| | 01 | 0011 | 0111 | 1100 | 1000 |
| | 10 | 1110 | 1101 | 0001 | 0010 |
| | 11 | 1100 | 1000 | 0011 | 0111 |

**Table 4.6:** Binary output values $\{I_{bb,3}(n),\ I_{bb,2}(n),\ I_{bb,1}(n),\ I_{bb,0}(n)\}$ of the carrier mixer.

The truth table translates into the following logical operations between the different sign and magnitude bits:

$$\begin{aligned} I_{bb,3}(n) &= I_1(n) \oplus K_1(n) \\ I_{bb,2}(n) &= I_{bb,3}(n) \oplus (I_0(n) \cdot K_0(n)) \\ I_{bb,1}(n) &= I_{bb,2}(n) \oplus I_0(n) \oplus K_0(n) \\ I_{bb,0}(n) &= \overline{I_{bb,2}(n) \oplus K_0(n)} \end{aligned} \tag{4.10}$$

By taking advantage of the redundancy and storing intermediate results, the complete carrier removal is operated with 12 operations. Following the carrier mixing operations, the code removal simply consists in a sign inversion, respectively non inversion, which translates into an exclusive OR between the code $c^x(n)$ and all the respective signal bits:

$$I_m^x(n) = c^x(n) \oplus I_{bb,m}(n) \tag{4.11}$$

where $\quad x \in \{E,\ P,\ L\}$.

The complete code removal is operated with 24 operations. From each of the $M$ words obtained with Equation 4.11, the sum $S_m^x$ is calculated by means of a LUT and summed up with the previous ones in order to form the final accumulation results expressed as:

$$I^x = -6 \cdot \sum S_3^x + 3 \cdot \sum S_2^x + 2 \cdot \sum S_1^x + \sum S_0^x \tag{4.12}$$

In addition to unavoidable load and store operations, Table 4.7 provides a rough estimation of the amount of operations per second necessary to process $N_c$ satellite channels in the base-band (without carrier and code generation). It is assumed here that the processor manipulates data words of $N_b$ bits.

| | # bitwise operations | # LUT accesses | # additions |
|---|---|---|---|
| Carrier mixing | $12 \cdot f_s \cdot N_c/N_b$ | - | - |
| Code mixing | $24 \cdot f_s \cdot N_c/N_b$ | - | - |
| Accumulation | - | $24 \cdot f_s \cdot N_c/N_b$ | $24 \cdot f_s \cdot N_c/N_b$ |
| Total | $36 \cdot f_s \cdot N_c/N_b$ | $24 \cdot f_s \cdot N_c/N_b$ | $24 \cdot f_s \cdot N_c/N_b$ |

**Table 4.7:** Amount of operations per second involved in the carrier mixing, the code mixing and the accumulation.

As compared to the classical bitwise processing of Subsection 4.1.4, the distributed arithmetic may require more logical operations for the carrier and code mixing. On the other hand, no additional stage is needed to convert the data from the bitwise into the integer representation, making this solution a prefect compromise between both approaches. In the case of the proposed 2-bit data configuration, the distributed arithmetic lowers the complexity by almost a factor two. It becomes even more efficient for higher signal bit-depths setups, as the complexity grows almost proportionally with the data bit quantization (cf. Equation 4.12), while it increases exponentially in the standard implementation (cf. Equation 4.3).

## 4.2 Carrier generation

The generation of a complex carrier within the receiver is requested to perform the Doppler removal. However, the implementation of a conventional NCO in software makes the real-time carrier generation computationally too expensive and not adapted. As compilers' trigonometric functions or Taylor series decompositions for the sine or cosine are also too much time consuming, new approaches have to be specifically developed.

### 4.2.1 Off-line carrier generation

The off-line generation consists in pre-computing and storing a set of carrier frequency candidates instead of generating them in real-time. As it would require a huge amount of memory to store all the possible frequencies, the values are recorded at the system sampling frequency on a coarse frequency grid and with a zero initial phase. The limited number of available carrier frequencies introduces an additional mismatch $\delta f$ in the Doppler removal process which causes a correlation loss. This way, Equation 2.26 becomes [Led03]:

$$\begin{aligned}(I^x)' &\approx a \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot (\Delta f + \delta f) \cdot T_{int})}{\pi \cdot (\Delta f + \delta f) \cdot T_{int}} \\ &\quad \cdot \cos(\pi \cdot (\Delta f + \delta f) \cdot T_{int} + \Delta\phi) \\ (Q^x)' &\approx a \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot (\Delta f + \delta f) \cdot T_{int})}{\pi \cdot (\Delta f + \delta f) \cdot T_{int}} \\ &\quad \cdot \sin(\pi \cdot (\Delta f + \delta f) \cdot T_{int} + \Delta\phi)\end{aligned} \tag{4.13}$$

where $\Delta f$ is the error between the real and the estimated frequency [Hz];
$\delta f$ is the error introduced by the frequency grid [Hz].

However, since the frequency mismatch $\delta f$ is known, the approximated results $(I^x)'$ and $(Q^x)'$ can be corrected in order to recreate an estimation $\widetilde{I^x}$ and $\widetilde{Q^x}$ of the original accumulated values $I^x$ and $Q^x$, as follows:

$$\begin{aligned}\widetilde{I^x} &= (I^x)' \cdot \cos(\pi \cdot \delta f \cdot T_{int}) + (Q^x)' \cdot \sin(\pi \cdot \delta f \cdot T_{int}) \\ \widetilde{Q^x} &= (-I^x)' \cdot \sin(\pi \cdot \delta f \cdot T_{int}) + (Q^x)' \cdot \cos(\pi \cdot \delta f \cdot T_{int})\end{aligned} \tag{4.14}$$

These operations of rotation are performed only once per integration. The development of Equation 4.14 leads to:

$$\begin{aligned}\widetilde{I^x} &\approx a \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot (\Delta f + \delta f) \cdot T_{int})}{\pi \cdot (\Delta f + \delta f) \cdot T_{int}} \cdot \cos(\pi \cdot \Delta f \cdot T_{int} + \Delta\phi) \\ \widetilde{Q^x} &\approx a \cdot N_s \cdot d \cdot R^x(\Delta\tau) \cdot \frac{\sin(\pi \cdot (\Delta f + \delta f) \cdot T_{int})}{\pi \cdot (\Delta f + \delta f) \cdot T_{int}} \cdot \sin(\pi \cdot \Delta f \cdot T_{int} + \Delta\phi)\end{aligned} \tag{4.15}$$

The comparison between Equation 2.26 and Equation 4.15 shows an additional offset $\delta f$ in the quotient term which acts as an attenuating factor. Consequently, the use of a restricted set of carrier frequencies introduces more attenuation than the rigorous method, since the frequency error due to the table quantization $\delta f$ will be generally larger than the tracking error $\Delta f$. However, if the frequency quantization is small enough relative to the integration time, the extra attenuation becomes negligible. For example, if a grid spacing of 150 Hz is selected for 1 ms integration time, $|\delta f| < 75$ Hz and the additional loss is contained within 0.08 dB [Led03].

The stored carrier sequences length must be equal to the number of samples contained in one integration period. Consequently longer integration times require longer sequences, but also finer grid spacing, so increasing the memory requirements proportionally to the square of the coherent integration time. Table 4.8 estimates the memory requirements, assuming a sampling frequency $f_s = 4.092$ MHz and a $\pm 5$ kHz Doppler range.

| Integration time | Frequency grid spacing | # frequency bins | Memory |
|---|---|---|---|
| 1 ms | 150 Hz | 67 | 548 kB |
| 10 ms | 15 Hz | 667 | 54.8 MB |

**Table 4.8:** Memory requirements for storing 1 and 10 ms complex carrier sequences covering 10 kHz frequency range.

In order to reduce the length of the stored sequences, a set of initial phases can be computed for each carrier candidate. By providing phase alignment capabilities, it becomes possible to rebuild a complete continuous waveform by juxtaposing several basic stored sequences [Nor04]. This way, the carrier sequences length becomes quasi independent of the integration time and can be selected with respect to the available memory space. The memory organization takes the form illustrated in Figure 4.6.

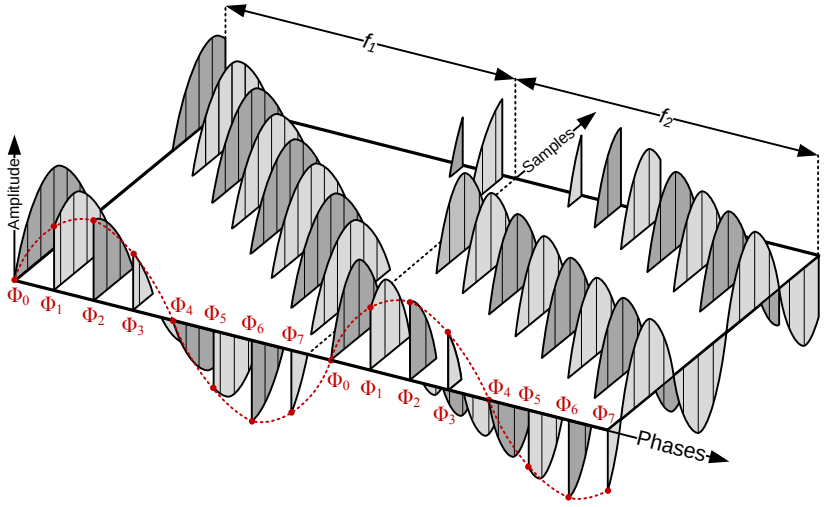


**Figure 4.6:** Pre-computed set of carrier frequencies ($f_1$ and $f_2$) and their respective eight initial phases $\Phi_0 - \Phi_7$ [Nor04].

Typically carrier sequences length of 512 points are pre-computed with eight different phases each. Table 4.9 estimates the memory requirements, assuming a ±5 kHz Doppler range.

| Integration time | Frequency grid spacing | # frequency bins | Memory |
|---|---|---|---|
| 1 ms | 150 Hz | 67 | 548 kB |
| 10 ms | 15 Hz | 667 | 5.48 MB |

**Table 4.9:** Memory requirements for storing 1 and 10 ms carrier sequences covering 10 kHz frequency range.

The pre-generation of the carrier is a very efficient alternative to the power-hungry real-time generation. It is very popular and many implementations proposed in the literature rely on it (see e.g., [Cha07] or [Led03]). However, it still suffers from its large memory requirements and lack of flexibility. For example, in case of strong oscillator drifts such as described in Section 3.3, the whole Doppler space can easily shift a few kilohertz away, making obsolete the set of pre-computed frequencies.

### 4.2.2 Single frequency carrier generation

The algorithm is implemented as a look-up table containing one single pre-generated frequency. The objective is to perform the carrier removal concurrently to all received satellite signals, once for all and with this same frequency. Considering a Doppler range of ±5 kHz, the frequency error most likely results in unacceptable loss levels. To overcome this, the integration period $T_{int}$ is split into $U$ sub-intervals for which the partial accumulations $(I_u^x)'$ and $(Q_u^x)'$ are computed and rotated proportionally to the frequency mismatch $\delta f$. The estimation $\widetilde{I^x}$ and $\widetilde{Q^x}$ of the original accumulated values $I^x$ and $Q^x$ can be expressed as follows [Pet08]:

$$\begin{aligned}
\widetilde{I^x} &= \sum_{u=1}^{U} \left( (I_u^x)' \cdot \cos(\pi \cdot \delta f \cdot \frac{T_{int} \cdot (2 \cdot u - 1)}{M}) \right) \\
&+ \sum_{u=1}^{U} \left( (Q_u^x)' \cdot \sin(\pi \cdot \delta f \cdot \frac{T_{int} \cdot (2 \cdot u - 1)}{M}) \right) \\
\widetilde{Q^x} &= \sum_{u=1}^{U} \left( (-I_u^x)' \cdot \sin(\pi \cdot \delta f \cdot \frac{T_{int} \cdot (2 \cdot u - 1)}{M}) \right) \\
&+ \sum_{u=1}^{U} \left( (Q_u^x)' \cdot \cos(\pi \cdot \delta f \cdot \frac{T_{int} \cdot (2 \cdot u - 1)}{M}) \right)
\end{aligned} \tag{4.16}$$

Recursively applying the phase compensation for all the $U$ sub-intervals mitigates the effect of the large frequency error and, with a careful selection of $U$, the attenuation factor can be limited to reasonable values. As the Doppler removal only needs to be performed once for all satellites, in a single mixer, the complexity is significantly reduced. However, the algorithm implementation remains difficult and necessitates some trade-offs that further increases the mean power loss [Pet06]. Due to these practical issues, the off-line carrier generation is generally preferred for its simplicity.

## 4.3 Code generation

The generation of the code replicas within the receiver is necessary to perform the correlation with the incoming signal. However, the implementation of a conventional NCO in software makes the real-time code generation computationally too expensive and new approaches have to be specifically developed.

### 4.3.1 Off-line code generation

Identically to the off-line carrier generation, this method consists in pre-computing all the PRN codes and storing them in memory. Each sequence is generated at the nominal code rate of 1.023 MHz, with no Doppler shift, and sampled at the system frequency. The consequence of the zero Doppler shift assumption is a small correlation power loss, which can be neglected if the Doppler range is contained within ±5 kHz. The memory also includes a selection of $m$ code sequences with different sampling offsets $t_{off}$ in order to match as close as possible the phase of the incoming signal [Led03].

$$t_{off}(l) = \frac{l}{f_s \cdot m} \; [s] \tag{4.17}$$

where $l$ is an integer defined as $1 \leq l \leq m$.

The different timing relationships are illustrated in Figure 4.7. The offsets grid spacing is chosen to be large enough to guarantee sufficient timing resolution for the tracking while keeping a reasonable table size. The higher the sampling frequency is, the lesser offsets are required and typically, for a sampling rate $f_s = 4.092$ MHz, less than 20 different offsets are stored, such as to provide code alignment capabilities of a few meters.

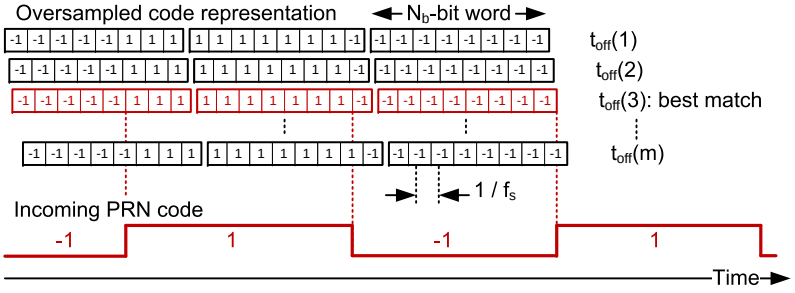


**Figure 4.7:** Pre-computed set of PRN codes sampled with $m$ different initial phases.

As the code value is quantized with one bit only, it would be too much memory consuming to store each sample separately as an integer. Consequently, consecutive samples are regrouped into words of 8, 16, 32 or 64 bits and stored.

However, as the incoming signal code phase is random, the beginning of the first code chip is most likely not synchronized with the beginning of a word and may occur anywhere within it. This can be solved either by storing all the possible phases in the memory or by shifting in real-time the code appropriately, as illustrated in Figure 4.8.

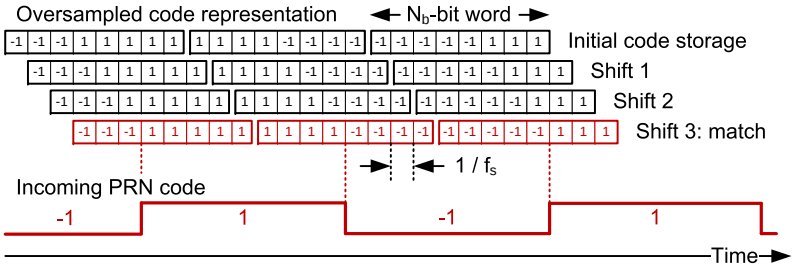


**Figure 4.8:** Successive logical shifts of the stored PRN code sequence to match the code phase of the incoming signal.

While the first solution increases the memory requirements proportionally to the word size $N_b$, the second necessitates further data processing in function of the phase mismatch (from 0 to $N_b/2$ logical shifts to be operated for each single word). The same also applies for the generation of the E and L code replicas. Both can be pre-generated and stored individually or can be real-time computed by shifting the reference sequence with the appropriate number of samples (which also limits the time-delay resolution to a multiple of one sampling period). Assuming a sampling frequency $f_s = 4.092$ MHz, the memory requirements for storing the 32 PRN codes separately for the E, P, and L replicas are presented in Table 4.10.

| Synchronization | # code offsets | # code shifts | Memory |
|---|---|---|---|
| Logical shift | 20 | 1 | 1 MB |
| Full storage | 20 | $N_b$ | $N_b$ MB |

**Table 4.10:** Memory requirements for storing the 32 PRN codes, separately for the E, P, and L replicas.

The off-line generation of the code is very popular and overcomes the high computational load inherent to the real-time generation. However, it presents the same drawbacks as the off-line carrier generation (i.e. large memory requirements up to several tens of megabytes and lack of flexibility). Furthermore, assuming zero Doppler shift on the code may become an issue for long integration times or in case of strong oscillator drifts such as described Section 3.3.

## 4.4 Summary

Chapter 4 reviews the different strategies proposed in the literature for lowering the complexity of the base-band operations. Many solutions rely on the use of specific CPU instructions schemes for improving the efficiency of the processing at the cost of a restricted portability of the code. On the other hand, the use of bitwise processing or the proposed distributed arithmetic algorithm still suffer from lack of flexibility. They require the front-end output stream to be specifically formatted, while any change in the signals structure implies a significant adaptation of the code and impacts the complexity. The synthesis of the carrier and code replicas also constitute one of the main bottlenecks of the software implementation. Since the computational load of generating them in real-time is too high, both are generally computed off-line and stored in memory. This may become an issue in case of strong oscillator drifts, since the whole Doppler space can easily shift a few kilohertz away, making obsolete the set of pre-computed carrier and code sequences.

# Chapter 5

# New architecture of a software receiver

The feasibility of building a software receiver operating in real-time was demonstrated in the literature. However, most of the proposed solutions rely on the same well known principles, mainly based on the use of SIMD and the pre-generation of the local signal replicas, and are tied to specific configurations that severely limit their portability. Consequently, in order to implement a software receiver operating in real-time on different platforms and accommodating various signals configurations, there is a need to develop a new architecture combining both flexibility and efficiency.

## 5.1 General concept

The fundamental concept of the proposed receiver architecture is derived from the restricted frequencies range of the carrier internally generated. The local carrier replicas are needed in the receiver to perform the down-conversion of the incoming satellites signals to base-band by removing the respective residual frequencies. These consist in the IF, given by the RF front-end architecture and common to all the satellite channels, plus the Doppler frequency $f_d$, due to the relative motion between the satellite and the user, which is specific to each satellite channel. The down conversion is generally done in the receiver by removing simultaneously both the IF and $f_d$ in a single mixer, requiring the frequency to be internally synthesized at a few or several megahertz ($f = \text{IF} + f_d$), as illustrated in Figure 5.1.

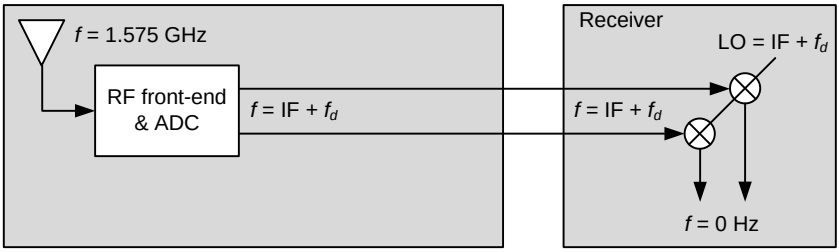


**Figure 5.1:** Traditional carrier down-conversion where both IF and residual Doppler are removed at once, using a single mixer.

By introducing an intermediate stage in-between the RF front-end and the receiver, the down-conversion can be split into two distinct steps, as illustrated in the Figure 5.2. The signal is first pre-converted to the residual Doppler frequency by removing the IF of a few megahertz in a top-level mixer common to all the satellite channels. Then, in a second stage, the signal is further down-converted to base-band by removing the residual Doppler frequency $f_d$ of a few kilohertz.

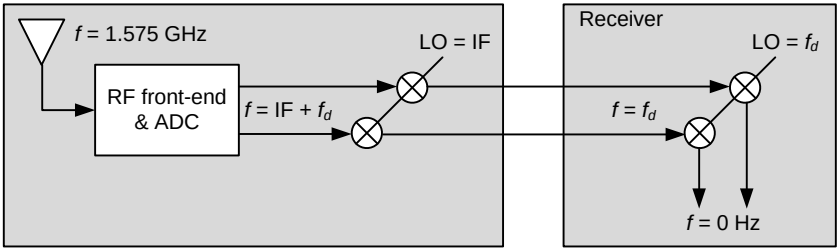


**Figure 5.2:** Two-step carrier down-conversion where the IF and the residual Doppler are removed separately.

The progressive removal of the carrier frequency relaxes the constraints on the second mixer, as the residual Doppler frequencies to be internally synthesized are now in the range of a few kilohertz only, instead of a few megahertz. This property can be largely exploited to simplify all the carrier related operations, but it also impacts positively the complexity of all the subsequent processing.

## 5.2 Base-band pre-processing

The carrier IF removal is achieved by means of a base-band pre-processing stage, introduced in between the RF front-end unit and the receiver. Since the IF depends on the RF front-end architecture only, the frequency translation can operate concurrently for all the satellite channels in the following ways:

- using a zero IF front-end [Raz98] which directly provides the digital stream modulated at the residual Doppler frequency;
- implementing a digital IF mixer in hardware within the RF front-end, in order to provide the receiver with a digital stream modulated at the residual Doppler frequency;
- implementing a digital IF mixer in software within the receiver, in order to provide the satellite channels with a digital stream modulated at the residual Doppler frequency.

The oscillator inaccuracy affects the carrier down-conversion and translates into an equivalent Doppler shift common to all the satellite channels. However, this effect can be compensated by adjusting the local frequency of the IF mixer accordingly (although this does not alleviate the additional Doppler on the code frequency which still needs to be further compensated). Therefore, we assume from now that the incoming signal distributed into the different receiver satellite channels is modulated by an absolute residual Doppler frequency within ±5 kHz.

## 5.3 Base-band processing

### 5.3.1 Batch processing applied to carrier removal

#### Real-time carrier generation

Let us consider the generation of the local carrier frequency by means of a conventional NCO. From Equation 2.18, the number of clock cycles $N_{clk}$ needed to achieve one complete carrier period (i.e. when the accumulated phase overflows the accumulator capacity), is given by:

$$N_{clk} = \frac{2^W}{Inc} = \frac{f_s}{f_d} \tag{5.1}$$

We make hereafter the distinction between a carrier phase, defined within one carrier period, and a carrier interval which repeats and extends over one complete integration period. Assuming a carrier quantized with $M$ bits that fix $2^{M+1}$ different phases, the average number of samples contained in each carrier interval can be expressed as:

$$\overline{a} = \frac{N_{clk}}{2^{M+1}} = \frac{f_s}{2^{M+1} \cdot f_d} \tag{5.2}$$

Considering the residual Doppler range of ±5 kHz, Equation 5.2 typically results in a hundred or thousand of samples per carrier interval. This order of magnitude is to be compared with a NCO operating around the IF of several megahertz and where a single carrier period is achieved within a few clock cycles only. Thus, when generating a frequency of a few kilohertz or less, the NCO phase accumulator increases very slowly and the rate at which the carrier magnitude changes is very low as compared to the sampling frequency. Since the carrier value keeps constant during several hundreds of clock cycles and can therefore be easily predicted over time, there is no more need to maintain the phase accumulator up to date on a per sample basis. The structure of the NCO can thus be simplified, in order to operate it at a much lower frequency, by predicting the sample times at which the carrier transitions occur. In that sense, the initial NCO phase accumulator is transformed into a samples accumulator $a_j$ incremented by the average number of samples $\overline{a}$ per carrier interval $j$:

$$\begin{aligned} a_0 &= 0 \\ a_j &= (j-1) \cdot \overline{a} + a_{off} \qquad\qquad 1 \leq j < J \\ a_J &= N_s \end{aligned} \tag{5.3}$$

where $a_{off}$ is the accumulator offset corresponding to the length of the first carrier interval in samples ($a_{off} \leq \overline{a}$);
$J$ is the number of carrier intervals per integration period.

The carrier accumulator is operated at a rate proportional to the Doppler frequency and requires now $J$ iterations instead of $N_s$. The samples boundaries of the $J$ carrier intervals are provided by the sequence $A_j$ given by:

$$A_j = \lfloor a_j \rfloor \tag{5.4}$$

where $\lfloor x \rfloor$ denotes the *floor(x)* function that rounds the value of $x$ towards the nearest smallest integer.

The carrier accumulator principle is illustrated in Figure 5.3.

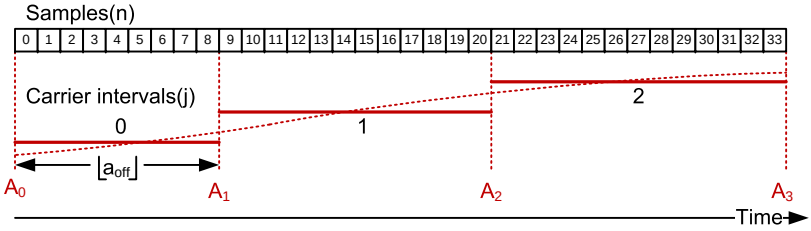


**Figure 5.3:** Samples boundaries $A_j$ of the carrier intervals.

Since the ratio of the carrier interval length $\overline{a}$ and the number of samples $N_s$ per integration most likely results in a non-integer value, the last interval, respectively the first one, may overlap two consecutive integration periods $T_1$ and $T_2$. The waveform continuity through the consecutive integration periods is kept by controlling the phase of the generated carrier, by means of $a_{off}(T_2)$ in Equation 5.3. The principle is illustrated in Figure 5.4.

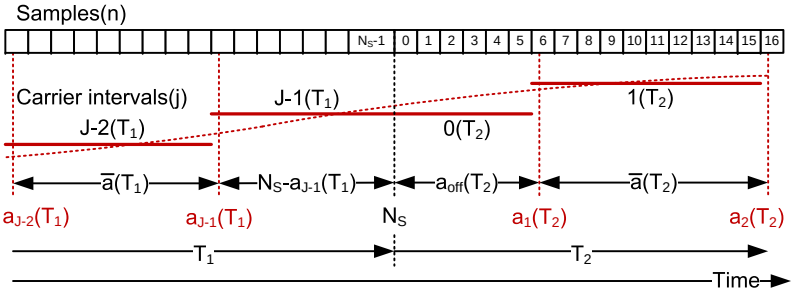


**Figure 5.4:** $J^{th}$ carrier interval overlapping two consecutive integration periods $T_1$ and $T_2$. The phase continuity is maintained by adjusting the length $a_{off}(T_2)$ of the first carrier interval of $T_2$.

If the end of the integration period $T_1$ coincides with the end of the $J^{th}$ carrier interval, then the carrier of $T_2$ is generated with a zero initial phase and $a_{off}(T_2) = \overline{a}(T_2)$. On the other hand, if a carrier interval overlaps two consecutive integration periods, then the carrier of $T_2$ is generated with a phase offset and the length of the first carrier interval becomes:

$$a_{off}(T_2) = \overline{a}(T_2) \cdot \left(1 - \frac{N_s - a_{J-1}(T_1)}{\overline{a}(T_1)}\right) \tag{5.5}$$

### Carrier mixing

The base-band demodulation can be mathematically expressed from Equation 2.20 and Equation 2.23 as:

$$\begin{aligned} I^x &= \sum_{n=0}^{N_s-1} c^x(n) \cdot \left[ I(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) + Q(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \right] \\ Q^x &= \sum_{n=0}^{N_s-1} c^x(n) \cdot \left[ Q(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) - I(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \right] \end{aligned} \tag{5.6}$$

Practically the complex carrier is quantized with $M$ bits and consists in a waveform that takes any of the $2^M$ discrete levels associated to the $2^{M+1}$ carrier phases (c.f. Subsection 2.3.1). When generated with a few kilohertz or less, the carrier remains unchanged during many clock cycles as defined in Equation 5.2, and many incoming samples $I(n)$ and $Q(n)$ are multiplied by the same value in the base-band down-conversion. The distributive law can be exploited and, instead of multiplying each data individually, samples appertaining to the same carrier interval are first accumulated into a batch (hereafter referred to as a partial sum) that is multiplied only once with the respective carrier magnitude. This way Equation 5.6 becomes:

$$\begin{aligned} I^x &= \sum_{j=0}^{J-1} \cos(j) \cdot \left( \sum_{n=A_j}^{A_{j+1}-1} c^x(n) \cdot I(n) \right) \\ &+ \sum_{j=0}^{J-1} \sin(j) \cdot \left( \sum_{n=A_j}^{A_{j+1}-1} c^x(n) \cdot Q(n) \right) \\ Q^x &= \sum_{j=0}^{J-1} \cos(j) \cdot \left( \sum_{n=A_j}^{A_{j+1}-1} c^x(n) \cdot Q(n) \right) \\ &- \sum_{j=0}^{J-1} \sin(j) \cdot \left( \sum_{n=A_j}^{A_{j+1}-1} c^x(n) \cdot I(n) \right) \end{aligned} \tag{5.7}$$

where $\sin(j)$ and $\cos(j)$ denote the complex carrier magnitude taken during the carrier interval $j$.

The data $I(n)$ and $Q(n)$ are first multiplied with the three code replicas $c^x(n)$ and pre-accumulated into partial sums, accordingly to the carrier interval boundaries defined by the sequence $A_j$ in Equation 5.3. The $J$ partial sums are then multiplied with the respective carrier magnitudes $\sin(j)$ and $\cos(j)$, and summed up to form the definitive correlation results $I^x$ and $Q^x$. The carrier removal is now performed on each partial sum, instead of each sample, at a rate proportional to the residual Doppler frequency. The data are said to be batch processed. This translates into a redistribution of the conventional base-band architecture, since the code removal intervenes now first, as illustrated in Figure 5.5.

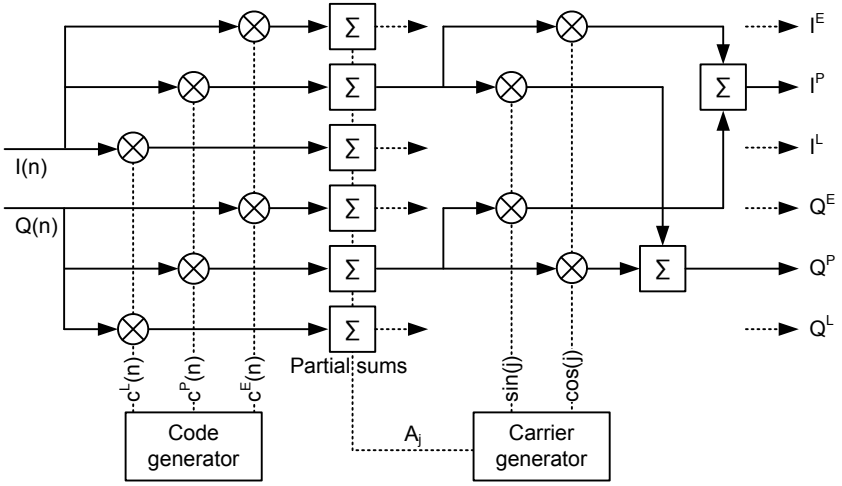


**Figure 5.5:** Base-band architecture with batch processing applied to carrier removal. Carrier and code operations order is swapped as compared to a traditional implementation.

Thanks to the carrier periodicity, every $2^{(M+1)th}$ partial sum is multiplied by the same carrier magnitude. Consequently, instead of performing $J$ multiplications, all the $2^{(M+1)th}$ partial sums are first regrouped per phase and summed up to be multiplied at once at the end of each integration period. This way, the data throughput is further decreased from the Doppler frequency to the number of carrier phases, quasi suppressing all the multiplications involved in the carrier removal process. This makes the carrier batch processing well fitted for a software receiver, as the complexity of the carrier mixing is no longer sampling frequency dependent, thus providing a maximum of flexibility for the implementation.

### 5.3.2 Batch processing applied to code removal

**Real-time code generation**

Let us consider the generation of the local code replica by means of a NCO. From Equation 2.18, the average number of samples contained in each chip (or the number of clock cycles needed to achieve one complete chip period) is given by:

$$\overline{b} = \frac{2^W}{Inc} = \frac{f_s}{f_c} \tag{5.8}$$

where $f_c$ is the chipping rate of the CA code.

Considering a sampling frequency $f_s = 4.092$ MHz, approximately four samples on average are contained in each chip. Thus, identically to the carrier generation, the goal is to lower the operating frequency of the code NCO by predicting the sample times at which the chip transitions occur. In that sense, the initial NCO phase accumulator is transformed into a samples accumulator $b_k$ incremented by the average number of samples $\overline{b}$ per chip $k$:

$$b_k = k \cdot \overline{b} + b_{off} \qquad 0 \leq k < K + 1 = \left\lceil \frac{N_s - b_{off}}{\overline{b}} \right\rceil + 1 \tag{5.9}$$

where $b_{off}$ is the accumulator offset corresponding to the length of the first incomplete chip in samples ($b_{off} < \overline{b}$);
$K$ is the number of chips processed per integration period.

The code accumulator is now operated at a rate proportional to the code frequency and requires $K$ iterations instead of $N_s$. The samples boundaries of the $K$ different chips are provided by the sequence $B_k$ defined as:

$$B_k = \lfloor b_k \rfloor \tag{5.10}$$

The generation of the different time-delayed code replicas can be emulated by shifting the boundaries of the sequence $B_k$ by the desired amount of samples $\Delta n$. This way and from the single accumulator of Equation 5.9, the chips boundaries of the E, P, and L code replicas are generated, with a time delay resolution of one sampling period $T_s$:

$$B_k^E = \lfloor b_k \rfloor \qquad B_k^P = \lfloor b_k \rfloor + \Delta n \qquad B_k^L = \lfloor b_k \rfloor + 2 \cdot \Delta n \tag{5.11}$$

The three sequences $B_k^x$ now define the chips boundaries of the E, P, and L code replicas, as illustrated in Figure 5.6. The integration periods of the P and L replicas are time shifted from $T_s \cdot \Delta n$ and $2 \cdot T_s \cdot \Delta n$ with respect to the E one and thus define the same chips sequence.

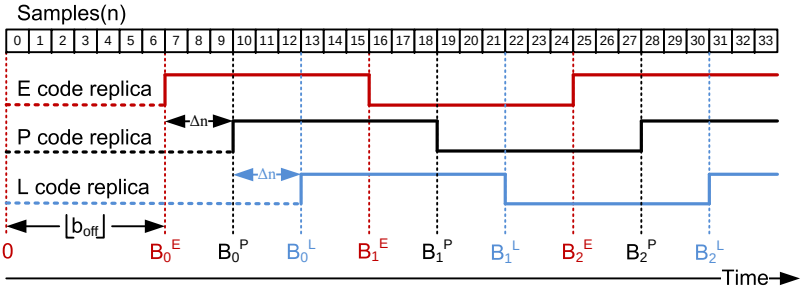


**Figure 5.6:** Chips boundaries of the E, P, and L code replicas.

The respective integration periods start at samples $B_0^x$, which coincide with the beginning of the first complete chip, and end at $B_K^x$, which coincide with the end of the last chip. The integrations are systematically run over an integer number of chips, thus guarantying that none of them overlap two consecutive integration periods $T_1$ and $T_2$. This condition simplifies the upcoming code mixing operation. The code waveform continuity is maintained by means of the accumulator offset $b_{off}$ in Equation 5.9 that is adjusted as follows:

$$b_{off}(T_2) = b_K(T_1) - N_s \tag{5.12}$$

Note that since the samples boundaries $B_K^x$ of the last chips equal or exceed $N_s$, dummy zero-valued samples may be needed to complete the integration.

### Code mixing

As defined in Equation 5.8, every chip value remains unchanged during several clock cycles and as many incoming samples $I(n)$ and $Q(n)$ are multiplied by the same chip during the code wipe-off process. Instead of multiplying each data individually, samples appertaining to the same chip are first accumulated into a batch (hereafter referred to as a chip sum) that is multiplied only once with the same chip value.

By applying batch-processing, Equation 5.6 becomes:

$$I^x = \sum_{k=0}^{K-1} C(k) \cdot \sum_{n=B_k^x}^{B_{k+1}^x - 1} \left[ I(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) + Q(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \right]$$

$$Q^x = \sum_{k=0}^{K-1} C(k) \cdot \sum_{n=B_k^x}^{B_{k+1}^x - 1} \left[ Q(n) \cdot \cos(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) - I(n) \cdot \sin(\widetilde{\omega} \cdot n \cdot T_s + \widetilde{\phi}) \right] \tag{5.13}$$

where $C(k)$ is the chip value taken during the chip sum $k$.

The data $I(n)$ and $Q(n)$ are first down-converted to base-band and pre-accumulated into chip sums, accordingly to the chips boundaries defined by the sequences $B_k^x$ in Equation 5.11. The $K$ chip sums are then multiplied with their respective code value $C(k)$ and summed up to form the definitive correlation results $I^x$ and $Q^x$. The resulting base-band architecture is illustrated in Figure 5.7.

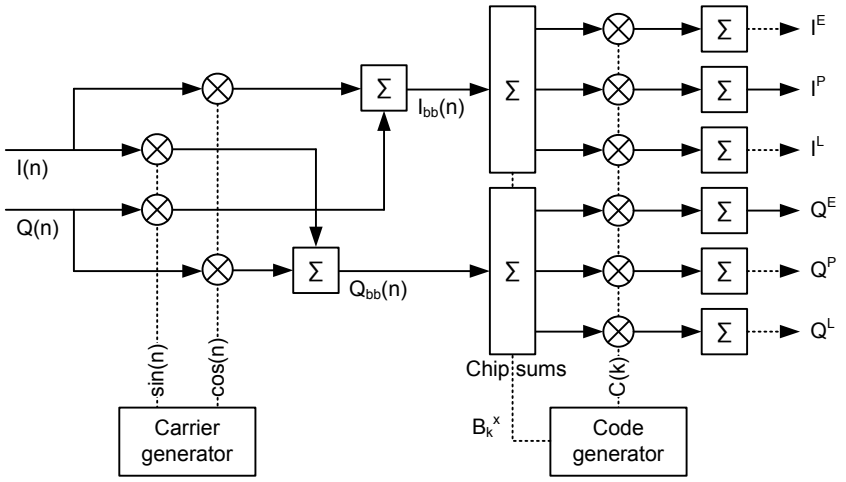


**Figure 5.7:** Base-band architecture with batch processing applied to code removal.

### 5.3.3 Proposed base-band architecture

The principle of batch processing can be further exploited and applied to both the carrier and code removal, by taking advantage of the architecture redistribution in Figure 5.5. The objective is to progressively reduce the data throughput from the sampling frequency to the code frequency, and finally to the number of carrier phases. This is achieved by first accumulating the incoming samples into complex chip sums, each one being multiplied by its corresponding chip value. The products are then summed up over the different carrier intervals in order to form the partial sums on which the carrier removal is further performed.

The operations described in Subsection 5.3.2 remain the same for the code removal. However, the batch processing applied to carrier removal is slightly modified, since the partial sums are now formed by summing up consecutive chip sums instead of samples. The average number of chip sums contained in each partial sum can be expressed as:

$$\overline{r} = \frac{\overline{a}}{\overline{b}} = \frac{f_c}{2^{M+1} \cdot f_d} \tag{5.14}$$

The carrier accumulator of Equation 5.3 is transformed in order to predict the chip times (instead of the sample times) where the carrier transitions occur. It consists now in a chips accumulator $r_j$ incremented by the average number of chips $\overline{r}$ per carrier interval $j$:

$$\begin{aligned} r_0 &= 0 \\ r_j &= (j-1) \cdot \overline{r} + r_{off} \qquad\qquad 0 < j < J \\ r_J &= \frac{b_K}{\overline{b}} \end{aligned} \tag{5.15}$$

where $r_{off}$ is the accumulator offset corresponding to the length of the first carrier interval in chips ($r_{off} \leq \overline{r}$);

Since the ratio of chips per carrier interval in Equation 5.14 most likely results in a non-integer value, the $J$ boundaries of the different intervals need to be approximated by rounding the sequence $r_j$ toward the nearest integer.

Since the integration periods of the the E, P, and L code replicas are time shifted with respect to each other, the chip sums boundaries of the intervals have to be re-estimated separately, by taking into account the respective delays $\Delta n$ and $2 \cdot \Delta n$ as follows:

$$\begin{aligned} R_0^E &= 0 & R_j^E &= \text{round}\,(r_j) \\ R_0^P &= 0 & R_j^P &= \text{round}\left(r_j - \frac{\Delta n}{\bar{b}}\right) \\ R_0^L &= 0 & R_j^L &= \text{round}\left(r_j - \frac{2 \cdot \Delta n}{\bar{b}}\right) \end{aligned} \tag{5.16}$$

The principle of the chips accumulator is illustrated in Figure 5.8, with the three sequences $R_j^x$ defining the carrier intervals boundaries in terms of chip sums for the respective E, P, and L code replicas. Note that the number of chip sums contained in each of the $j^{th}$ interval may differ for each of the replicas.

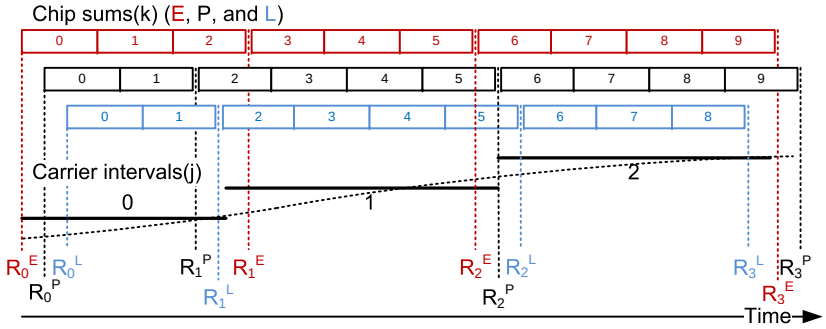


**Figure 5.8:** Carrier intervals boundaries of the three E, P, and L code replicas.

Since the number of carrier intervals per integration period most likely results in a non-integer value, the last interval, respectively the first one, may overlap two integration periods $T_1$ and $T_2$. The waveform continuity through the consecutive integration periods is kept by controlling the phase of the generated carrier, by means of $r_{off}(T_2)$ in Equation 5.15. The principle is illustrated in Figure 5.9.

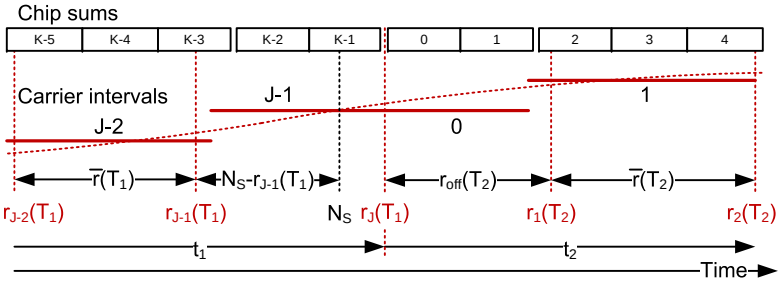


**Figure 5.9:** $J^{th}$ carrier interval overlapping two consecutive integration periods $T_1$ and $T_2$. The phase continuity is maintained by adjusting the length $r_{off}(T_2)$ of the first carrier interval of $T_2$.

If the end of the $K^{th}$ chip of the integration period $T_1$ coincides with the end of the $J^{th}$ carrier interval, then the carrier of $T_2$ is generated with a zero initial phase and $r_{off}(T_2) = \overline{r}(T_2)$. On the other hand, if a carrier interval overlaps two consecutive integration periods, which is most likely the case, then the carrier of $T_2$ is generated with a phase offset and the length of the first interval becomes:

$$r_{off}(T_2) = \overline{r}(T_2) \cdot \left(1 - \frac{r_J(T_1) - r_{J-1}(T_1)}{\overline{r}(T_1)}\right) \tag{5.17}$$

By introducing the batch processing for both the carrier and code removal, the base-band demodulation in Equation 5.6 becomes:

$$\begin{aligned} I^x &= \sum_{j=0}^{J-1} \cos(j) \cdot \left[ \sum_{k=R_j^x}^{R_{j+1}^x-1} C(k) \cdot \left( \sum_{n=B_k^x}^{B_{k+1}^x-1} I(n) \right) \right] \\ &+ \sum_{j=0}^{J-1} \sin(j) \cdot \left[ \sum_{k=R_j^x}^{R_{j+1}^x-1} C(k) \cdot \left( \sum_{n=B_k^x}^{B_{k+1}^x-1} Q(n) \right) \right] \\ Q^x &= \sum_{j=0}^{J-1} \cos(j) \cdot \left[ \sum_{k=R_j^x}^{R_{j+1}^x-1} C(k) \cdot \left( \sum_{n=B_k^x}^{B_{k+1}^x-1} Q(n) \right) \right] \\ &- \sum_{j=0}^{J-1} \sin(j) \cdot \left[ \sum_{k=R_j^x}^{R_{j+1}^x-1} C(k) \cdot \left( \sum_{n=B_k^x}^{B_{k+1}^x-1} I(n) \right) \right] \end{aligned} \tag{5.18}$$

The data $I(n)$ and $Q(n)$ are pre-accumulated into chip sums, accordingly to the chips boundaries defined by the sequences $B_k^x$ in Equation 5.11. The $K$ chip sums are then multiplied with their respective code value $C(k)$ and the products accumulated into partial sums, accordingly to the carrier interval boundaries defined by the sequences $R_j^x$ in Equation 5.16. The $J$ partial sums are distributed over the $2^{(M+1)}$ carrier phases, multiplied with the respective carrier magnitudes $\sin(j)$ and $\cos(j)$, and summed up to form the definitive correlation results $I^x$ and $Q^x$. The principle of batch processing applied to carrier and code removal is illustrated in Figure 5.10.

For each satellite channel the complex chip sums are now directly computed from the incoming samples as follows:

$$S(B_k^x : B_{k+1}^x - 1) = \sum_{n=B_k^x}^{B_{k+1}^x-1} (I(n) + i \cdot Q(n)) \tag{5.19}$$

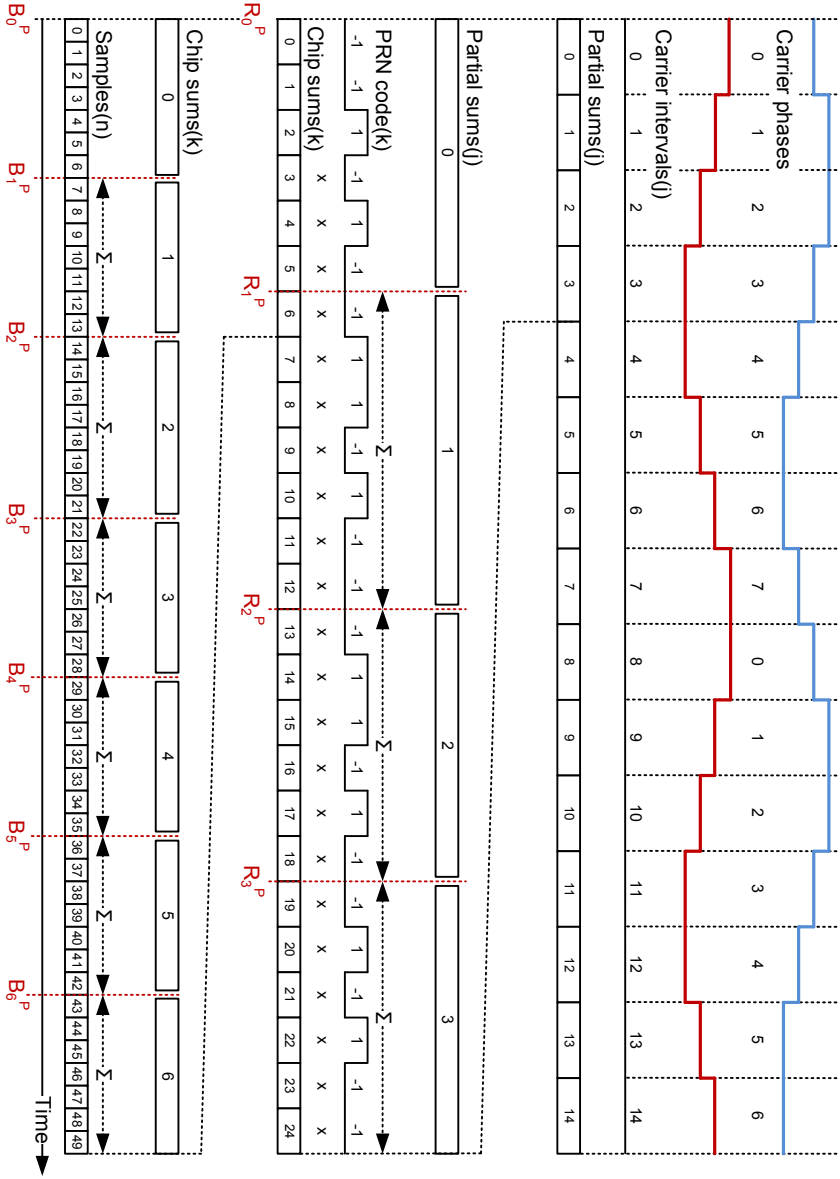


**Figure 5.10:** Batch processing applied to carrier and code operations [Bue09] and [Bue10].

They can be regrouped into a complex matrix of the size [$K$ x 3], where each row represents a chip $k$, and each column a code offset, respectively 0, $\Delta n$, and $2 \cdot \Delta n$. The matrix takes the form of Table 5.1.

| | | Code delay in samples (E, P, and L) | | |
|---|---|---|---|---|
| | | 0 | $\Delta n$ | $2 \cdot \Delta n$ |
| Chip index $k$ | 0 | $S(B_0^E{:}B_1^E\text{-}1)$ | $S(B_0^P{:}B_1^P\text{-}1)$ | $S(B_0^L{:}B_1^L\text{-}1)$ |
| | 1 | $S(B_1^E{:}B_2^E\text{-}1)$ | $S(B_1^P{:}B_2^P\text{-}1)$ | $S(B_1^L{:}B_2^L\text{-}1)$ |
| | 2 | $S(B_2^E{:}B_3^E\text{-}1)$ | $S(B_2^P{:}B_3^P\text{-}1)$ | $S(B_2^L{:}B_3^L\text{-}1)$ |
| | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| | K-1 | $S(B_{\text{K-1}}^E{:}B_K^E\text{-}1)$ | $S(B_{\text{K-1}}^P{:}B_K^P\text{-}1)$ | $S(B_{\text{K-1}}^L{:}B_K^L\text{-}1)$ |

**Table 5.1:** Chip sums matrix.

For a given sampling frequency, the chip sums size depends on the code frequency and is therefore specific to each satellite channel. However, although the number of samples per chip sum changes and is either floor($\bar{b}$) or ceil($\bar{b}$), the summation can, with negligible effects (see Subsection 6.2.1), always run over a constant samples interval defined as:

$$\overline{B} = \text{round}\left(\frac{f_s}{f_c}\right) = \text{round}(\bar{b}) \tag{5.20}$$

Assuming an unique summation interval $\overline{B}$, each chip sum is now computed as in Equation 5.21 and the matrix takes the form of Table 5.2.

$$S(B_k^x : B_k^x + \overline{B} - 1) = S_{B_k}^x = \sum_{n=B_k^x}^{B_k^x+\overline{B}-1} (I(n) + i \cdot Q(n)) \tag{5.21}$$

With the length of the summation interval set constant, many chip sums are commonly shared by the different satellite channels. Instead of recomputing separately all the chip sums for each satellite channel, the objective is to compute them once for all.

| | | Code delay in samples (E, P, and L) | | |
|---|---|---|---|---|
| | | 0 | $\Delta n$ | $2 \cdot \Delta n$ |
| Chip index $k$ | 0 | $S(B_0^E{:}B_0^E{+}\overline{B}\text{-}1)$ | $S(B_0^P{:}B_0^P{+}\overline{B}\text{-}1)$ | $S(B_0^L{:}B_0^L{+}\overline{B}\text{-}1)$ |
| | 1 | $S(B_1^E{:}B_1^E{+}\overline{B}\text{-}1)$ | $S(B_1^P{:}B_1^P{+}\overline{B}\text{-}1)$ | $S(B_1^L{:}B_1^L{+}\overline{B}\text{-}1)$ |
| | 2 | $S(B_2^E{:}B_2^E{+}\overline{B}\text{-}1)$ | $S(B_2^P{:}B_2^P{+}\overline{B}\text{-}1)$ | $S(B_2^L{:}B_2^L{+}\overline{B}\text{-}1)$ |
| | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| | K-1 | $S(B_{\text{K-1}}^E{:}B_{\text{K-1}}^E{+}\overline{B}\text{-}1)$ | $S(B_{\text{K-1}}^P{:}B_{\text{K-1}}^P{+}\overline{B}\text{-}1)$ | $S(B_{\text{K-1}}^L{:}B_{\text{K-1}}^L{+}\overline{B}\text{-}1)$ |

**Table 5.2:** Chip sums matrix with a constant summation interval of $\overline{B}$.

Consequently, for each of the incoming sample $I(n)$ and $Q(n)$, a complex chip sum $S_n$ is formed by summing up $\overline{B}$ consecutive data values, starting from the sample $n$ in question. The process takes place iteratively with two additions and subtractions per complex chip sum, instead of the $2 \cdot (\overline{B} - 1)$ additions originally required in Equation 5.21:

$$S_0 = \sum_{n=0}^{\overline{B}-1} (I(n) + i \cdot Q(n)) \tag{5.22}$$

$$S_{n+1} = S_n + I(n + \overline{B}) - I(n) + i \cdot [Q(n + \overline{B}) - Q(n)] \quad 0 \leq n < N_s - 1$$

All the so $N_s$ resulting chip sums are chronologically collected into a vector, common to all satellite channels and of the form of Table 5.3.

| | | |
|---|---|---|
| Sample index $n$ | 0 | $S_0{=}S(0{:}\overline{B}\text{-}1)$ |
| | 1 | $S_1{=}S(1{:}\overline{B})$ |
| | 2 | $S_2{=}S(2{:}\overline{B}{+}1)$ |
| | $\vdots$ | $\vdots$ |
| | $N_s$-1 | $S_{N_s-1}{=}S(N_s\text{-}1{:}N_s{+}\overline{B})$ |
| | $\vdots$ | 0 |

**Table 5.3:** Chip sums vector with a constant summation interval of $\overline{B}$.

The vector is addressed separately by each satellite channel, accordingly to the sequences $B_k^x$ that directly provide the respective indexes of the $3 \cdot K$ chip sums to select. The so formed E, P, and L chip sums sequences are respectively of the form:

$$\begin{array}{lllll} \{S_{B_0^E}, & S_{B_1^E}, & S_{B_2^E}, & \ldots & S_{B_{K-1}^E}\} \\ \{S_{B_0^P}, & S_{B_1^P}, & S_{B_2^P}, & \ldots & S_{B_{K-1}^P}\} \\ \{S_{B_0^L}, & S_{B_1^L}, & S_{B_2^L}, & \ldots & S_{B_{K-1}^L}\} \end{array} \tag{5.23}$$

Each sequence is multiplied point by point with the same PRN code $C(k)$ and the products are summed over the different carrier intervals, accordingly to the boundaries $R_j^x$ defined in Equation 5.16. The results are distributed over the $2^{(M+1)}$ carrier phases, multiplied by the respective carrier magnitudes $\sin(j)$ and $\cos(j)$, and rearranged in order to form the definitive correlation results $I^x$ and $Q^x$, [Bue09], and [Bue10]. This leads to the new base-band architecture depicted in Figure 5.11.

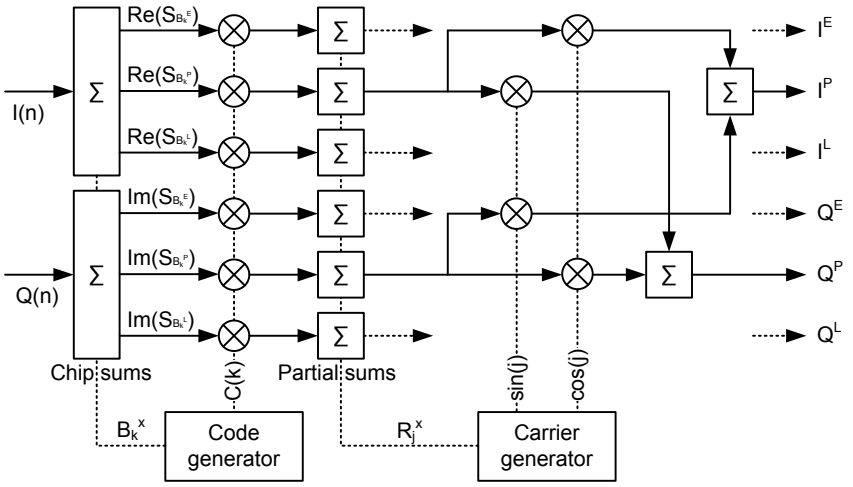


**Figure 5.11:** New base-band architecture based on batch processing.

## 5.4 Base-band algorithms

### 5.4.1 Acquisition algorithms

In order to make easier the coexistence of satellite channels in acquisition and in tracking mode, the objective is to re-use the structure of the base-band architecture for the acquisition. Accordingly to Equation 5.18:

- shifting the summation boundaries $B_k$ by $\Delta n$ is equivalent to delay the code replica by $\Delta n$ sample(s);
- shifting the chip $C(k)$ from $\Delta k$ is equivalent to delay the code replica by $\Delta k$ chip(s).

Consequently, multiplying consecutive time-delayed chip sums sequences with the same code vector corresponds to test consecutive code phases with one sample step size. In the same manner, for a given chip sums sequence, circular-shifting the code vector chip by chip corresponds to test consecutive code phases with one chip step size.

#### Parallel code search

The algorithm is based on the architecture described in Section 2.4.1, however it differs from it, since the FFT is not computed on the $N_s$ incoming samples, but on a chip sums sequence of length $K$. The latter is selected from the chip sums vector, accordingly to the chip boundaries $B_k$ of Equation 5.9, and is of the form:

$$\{S_{B_0}, \quad S_{B_1}, \quad S_{B_2}, \quad \dots \quad S_{B_{K-1}}\} \tag{5.24}$$

The local carrier is generated by means of Equation 5.15. Accordingly to the carrier intervals boundaries $R_j^E$, a sequence containing the $K$ carrier magnitude values, respectively associated to each chip sum, is produced and is of the form:

$$\{\cos(0) - i \cdot \sin(0), \quad \dots \quad \cos(K-1) - i \cdot \sin(K-1)\} \tag{5.25}$$

Both chip sums and carrier sequences are multiplied point by point and the result is left zero padded in order to form a complex vector which length $N$ is a power of two given by:

$$N = 2^{\lceil \log_2(K) \rceil} \tag{5.26}$$

The so formed sequence is transformed into the frequency domain by FFT and multiplied point by point with the FFT of the code (right zero padded in order to form a $N$-point vector). The result of the multiplication is transformed into the time domain by an inverse FFT and the absolute value computed for providing the correlation between the input signal and the PRN code. This allows testing all the possible code phases at once with one chip step size, for a given Doppler bin. The operation is reiterated with different carrier frequency vectors until the whole Doppler space is swept. In order to test different code phases with one sample step size, the whole operation is repeated by selecting a new chip sums sequence where the summation boundaries are shifted by one sample with respect to the previous one. The parallel code search architecture based on the batch processing is illustrated in Figure 5.12.

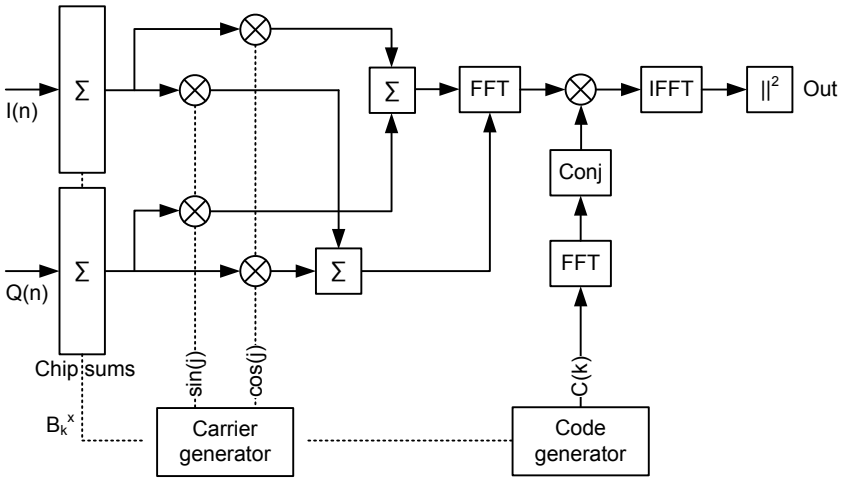


**Figure 5.12:** Batch processing applied to the parallel code search.

**Parallel frequency search**

The algorithm is based on the architecture described in Section 2.4.1, however it differs from it, since the pre-detection sums are not computed from the $N_s$ incoming samples but from a chip sums sequence of length $K$. The latter is selected from the chip sums vector, accordingly to the chip boundaries $B_k$ of Equation 5.9, and is of the form:

$$\{S_{B_0}, \quad S_{B_1}, \quad S_{B_2}, \quad \dots \quad S_{B_{K-1}}\} \tag{5.27}$$

The code vector is formed by repeating several times the original PRN code period to form a sequence of length $K$:

$$\{C(0), \quad C(1), \quad C(2), \quad \dots \quad C(\text{mod(K-1,1023)})\} \tag{5.28}$$

Both chip sums and code sequences are multiplied point by point to remove the code and the products are distributed over the $N_p$ pre-detection sums intervals. The boundaries of the latter are predicted by a pre-detection accumulator $v_i$, incremented by the average number of chip sums $\overline{v}$ per pre-detection sum:

$$v_i = i \cdot \overline{v} \qquad\qquad 0 \leq i < N_p \tag{5.29}$$

where $\overline{v}$ is given by:

$$\overline{v} = \frac{T_p \cdot f_s}{\overline{b}} = T_p \cdot f_c \tag{5.30}$$

The chip sums boundaries of the $N_p$ pre-detection sums are provided by the sequence $V_i$ defined as:

$$V_i = \text{round}(v_i) \tag{5.31}$$

The so formed pre-detection sequence is transformed into the frequency domain by FFT (zero padding is applied if needed in order to form a vector which length is a power of two). This allows testing all the possible Doppler offsets at once for a given code phase. In order to test different code phases with one chip step size, new partial correlations sequences are recomputed by circular-shifting the code vector chip by chip. To test different code phases with one sample step size, a new chip sums sequence is selected by shifting the summation boundaries by one sample with respect to the previous one.

The parallel frequency search architecture based on the batch processing is illustrated in Figure 5.13.

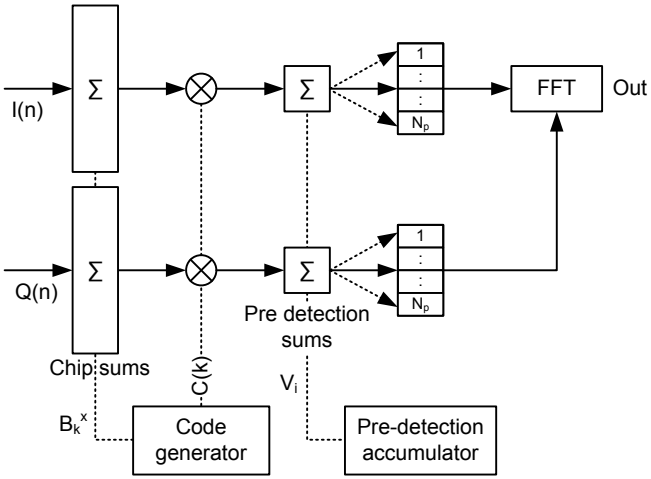


**Figure 5.13:** Batch processing applied to the parallel frequency search.

### 5.4.2 Tracking algorithms

The real-time generation of both the carrier and the code allows a straightforward implementation of conventional tracking algorithms.

## 5.5 Pseudoranges measurement

The code phase measurements consist in determining the exact phase of the locally generated code, at fixed time intervals, needed for further computing the pseudoranges. Each phase is composed of an integer part, given by the index of the chip generated at the time of measurement, and a fractional part given by the state of the code accumulator at the time of measurement. For all the satellite channels, the measurements are synchronized with the $N_s{}^{th}$ sample, as illustrated in Figure 5.14.

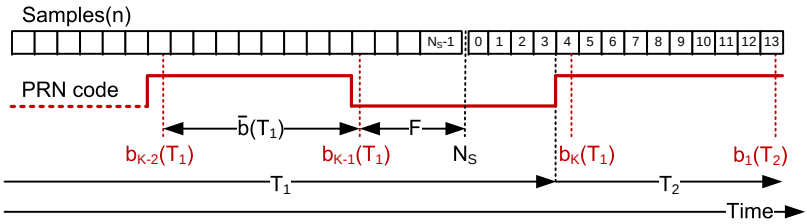


**Figure 5.14:** Code phase measurements synchronized with the $N_s^{th}$ sample. $F$ represents the fractional part of the chip generated at sample time $N_s$.

The state of the code accumulator is known at samples times $b_{K-1}$ and $b_K$ that generally do not coincide with $N_s$ (remember that the number of chips processed per integration period is round up to the next integer and that the boundaries of the last chip sum may exceed $N_s$). Consequently, interpolation may be needed to estimate the fractional part $F$ of the chip generated at sample time $N_s$, as follows:

$$F = \frac{\bar{b} + N_s - b_K(T_1)}{\bar{b}} \tag{5.32}$$

The fractional part $F$ of the chip is added to the integer part given by the index of the chip $K-1$ processed at sample time $N_s$. This finally forms a value in the range $[0; 1023[$ chips that is added to the code ambiguity, resolved by decoding the navigation message, transformed into a distance and further transmitted to the receiver PVT solution.

## 5.6 Summary

Chapter 5 introduces a completely new base-band architecture based on the concept of batch processing of the incoming samples. The basic idea consists in restricting the frequency of the carrier replica internally generated to a few kilohertz, so that its time evolution becomes very slow as compared to the sampling frequency. Since the carrier magnitude remains constant during hundreds or thousands of clock cycles, data appertaining to the same intervals are accumulated into batches that are multiplied only once by the corresponding carrier magnitude, instead of multiplying each data separately.

The same also applies for the code removal, where samples appertaining to the same chip are summed up and multiplied only once by the chip value. This results in a redistributed base-band architecture where the data throughput is progressively decreased and consequently the amount of operations involved is significantly reduced. The batch processing also simplifies the code and carrier synthesis, as they now consist in predicting the times where the chips, respectively the carrier intervals, transitions occur. Both the carrier and code generators operate at a low frequency, respectively proportional to the residual Doppler and the code rate, which allows the real-time generation of the signals waveforms at any desired frequency and with any phase. This makes any Doppler offset compensation possible (especially on the code as compared to pre-generated sequences that assume a zero frequency offset) and thus greatly simplifies the integration of the carrier and code blocks in the tracking loops.

# Chapter 6

# Performance of the new architecture

The theoretical performance, as well as the trade-offs and limitations of the proposed architecture are analyzed in this chapter. For each block of Figure 5.11, the complexity in terms of the amount of integer operations per second is computed. The ability of the receiver to accommodate some future GNSS signals is also discussed.

## 6.1 Performance of the batch-processing

### 6.1.1 Accumulation

The accumulation takes place in two different phases. The incoming complex samples are first summed up into complex chip sums that are iteratively computed with respectively two additions and subtractions. In addition to the unavoidable load and store operations, Table 6.1 provides a rough estimation of the amount of integer operations per second necessary to compute the complex chip sums.

| | # additions | # multiplications |
|---|---|---|
| Chip sums | $4 \cdot f_s$ | - |

**Table 6.1:** Amount of integer operations per second involved in the complex chip sums computation.

Note that the complex chips sums computation is performed independently of the number of satellite channels and represents an unavoidable basic load.

The second accumulation phase intervenes after the code removal, when the chip sums are summed up into partial sums. The accumulation is performed at the code rate $f_c$, in parallel for each of the E, P, and L signal components. In addition to the unavoidable load and store operations, Table 6.2 provides a rough estimation of the amount of integer operations per second necessary to compute the partial sums for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Partial sums | $6 \cdot f_c \cdot N_c$ | - |

**Table 6.2:** Amount of integer operations per second involved in the partial sums computation.

As compared to the traditional implementation in Table 3.2, the gain $G$ of the accumulation (chip sums and partial sums) in terms of the amount of operations can be roughly estimated as:

$$G \propto \frac{f_s}{\frac{2 \cdot f_s}{3 \cdot N_c} + f_c} \tag{6.1}$$

Assuming a sampling frequency $f_s = 8$ MHz and a 12 satellite channels configuration, $G \approx 5.5$. Note that the higher the sampling frequency is, the better the gain is too.

### 6.1.2 Real-time code generation

The code generation consists in predicting the samples boundaries of the different chips for the E, P, and L code replicas. It is implemented as a samples accumulator incremented by the average number of samples per chip, at a rate proportional to the code frequency $f_c$. The P and L sequences are derived from the accumulator by adding a constant offset $\Delta n$ and $2 \cdot \Delta n$ respectively. In addition to the unavoidable load and store operations, Table 6.3 provides a rough estimation of the amount of integer operations per second necessary to generate the code for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Code generation | $3 \cdot f_c \cdot N_c$ | - |

**Table 6.3:** Amount of integer operations per second involved in the code generation.

As compared to the traditional implementation where the NCO operates at the sampling frequency, the gain $G$ in terms of the amount of operations can be roughly estimated as:

$$G \propto \frac{f_s}{f_c} \tag{6.2}$$

Assuming a sampling frequency $f_s = 8$ MHz, $G \approx 8$.

As the complexity is no longer sampling frequency dependent, flexibility is achieved for the implementation and leads to an amount of operations which is now fully suitable for a real-time software receiver. Furthermore, as compared to the solution based on pre-computed and stored code sequences, the use of a NCO allows the continuous generation of the waveform at any desired frequency. This makes any Doppler offset compensation possible and also greatly simplifies the integration of the code block in the receiver, as it will easily accommodate traditional tracking schemes. Finally, the memory requirements are reduced to the strict minimum as only 1023 chips per code need to be stored.

### 6.1.3 Code mixing

The code mixing is performed by multiplying the E, P, and L complex chip sums sequences with the PRN code, chip by chip, at the code rate $f_c$. In addition to the unavoidable load and store operations, Table 6.4 provides a rough estimation of the amount of integer operations per second necessary to perform the code mixing for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Code mixing | - | $6 \cdot f_c \cdot N_c$ |

**Table 6.4:** Amount of integer operations per second involved in the carrier mixing.

The code removal is performed on each chip sum, instead of each sample, progressively decreasing the data throughput from the sampling to the code frequency. As compared to the traditional implementation, where the code mixing is performed at the sampling frequency, the gain $G$ in terms of the amount of operations can be roughly estimated as:

$$G \propto \frac{f_s}{f_c} \tag{6.3}$$

Assuming a sampling frequency $f_s = 8$ MHz, $G \approx 8$.

### 6.1.4 Real-time carrier generation

The carrier generation consists in predicting the chips boundaries of the different carrier intervals, for the E, P, and L signal components. It is implemented as a chips accumulator incremented by the average number of chips per carrier interval, at a rate proportional to the residual Doppler frequency $f_d$. The boundaries of the P and L sequences are derived from the accumulator by subtracting a constant offset proportional to the replicas delay $\Delta n$ and $2 \cdot \Delta n$ respectively. In addition to the unavoidable load and store operations, Table 6.5 provides a rough estimation of the amount of integer operations per second necessary to generate the carrier for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Carrier generation | $3 \cdot f_d \cdot 2^{M+1} \cdot N_c$ | - |

**Table 6.5:** Amount of integer operations per second involved in the carrier generation.

As compared to the traditional implementation where the NCO is operated at the sampling frequency, the gain $G$ in terms of the amount of operations can be roughly estimated as:

$$G \propto \frac{f_s}{3 \cdot f_d \cdot 2^{M+1}} \tag{6.4}$$

Assuming a sampling frequency $f_s = 8$ MHz and 3-bit carrier quantization, $G > 30$. The maximal performance is obtained when the residual Doppler frequency is quasi null and there is only one single batch to process. The computational burden of the real-time generation is drastically decreased.

The complexity of the carrier generation is no longer sampling frequency dependent and leads to an amount of operations which is now fully suitable for a real-time software receiver [Wae09b]. As compared to the solutions based on the pre-computed and stored carrier sequences, the use of a NCO allows the continuous generation of the exact waveform, at any desired frequency. This makes any Doppler offset compensation possible and also greatly simplifies the integration of the carrier block in the receiver as it will easily accommodate traditional tracking schemes. Finally, the memory requirements are reduced to the strict minimum and only $2^{M+1}$ carrier magnitudes need to be stored.

### 6.1.5 Carrier mixing

The complex chip sums are distributed over the carrier intervals, accordingly to the chips boundaries defined by the carrier generator, and accumulated over the $2^{M+1}$ carrier phases. The carrier mixing is performed at the end of each integration by multiplying each partial sum by its corresponding complex magnitude, and by finally recombining the in-phase and quadrature components accordingly to Equation 2.20. This is done in parallel for the three E, P, and L signal components. In addition to the unavoidable load and store operations, Table 6.6 provides a rough estimation of the amount of integer operations per second necessary to perform the carrier mixing for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Carrier mixing | $6 \cdot 2^{M+1} \cdot N_c/T_{int}$ | $12 \cdot 2^{M+1} \cdot N_c/T_{int}$ |

**Table 6.6:** Amount of integer operations per second involved in the carrier mixing.

The carrier removal is performed on each carrier phase, instead of each sample, reducing the data throughput from the sampling frequency to the number of carrier phases. As compared to the traditional implementation where the down-conversion is operated at the sampling frequency, the gain $G$ in terms of the amount of operations can be roughly estimated as:

$$G \propto \frac{f_s \cdot T_{int}}{3 \cdot 2^{M+1}} \tag{6.5}$$

Assuming a sampling frequency $f_s = 8$ MHz, a 3-bit carrier quantization and an integration time $T_{int} = 1$ ms, $G \approx 1.7 \cdot 10^2$. The computational burden of the carrier mixing is drastically decreased.

With respect to Table 3.2, the main improvement lies in the quasi absence of multiplications, making the carrier batch processing particularly well suited for a software implementation. The process becomes now dependent of the carrier quantization, but with a quasi negligible impact on the global performance.

Table 6.7 and Table 6.8 summarize the above estimations of the amount of integer operations per second necessary to generate the local code and carrier replicas, and to perform the correlation related operations for $N_c$ satellite channels.

| | # additions | # multiplications |
|---|---|---|
| Code generation | $3 \cdot f_c \cdot N_c$ | - |
| Carrier generation | $3 \cdot f_d \cdot 2^{M+1} \cdot N_c$ | - |

**Table 6.7:** Amount of integer operations per second involved in the real-time generation of the carrier and code.

| | # additions | # multiplications |
|---|---|---|
| Accumulation | $4 \cdot f_s + 6 \cdot f_c \cdot N_c$ | - |
| Code mixing | - | $6 \cdot f_c \cdot N_c$ |
| Carrier mixing | $6 \cdot 2^{M+1} \cdot N_c / T_{int}$ | $12 \cdot 2^{M+1} \cdot N_c / T_{int}$ |

**Table 6.8:** Amount of integer operations per second involved in the correlation related operations.

As compared to the complexity of the original base-band implementation presented in Table 3.2, the main improvement lies in the quasi independence of the overall complexity to the sampling frequency. The chip sums computation, intervening at the beginning of the processing chain, is the only operation performed at the sampling rate, thus allowing the receiver to accommodate high sampling frequencies without penalizing the computational load of the subsequent operations. This is highlighted in Table 6.9, which compares the amount of additions, respectively multiplications, involved in the two architectures for different sampling frequencies. A 12 satellite channels receiver configuration with 3-bit signal quantization and 1 ms integration time are assumed.

| | Sampling frequency $f_s$ | | |
|---|---|---|---|
| | 4.092 MHz | 8 MHz | 16 MHz |
| # additions | | | |
| Standard processing | $3.9 \cdot 10^8$ | $7.7 \cdot 10^8$ | $1.5 \cdot 10^9$ |
| Batch processing | $9 \cdot 10^7$ | $1 \cdot 10^8$ | $1.4 \cdot 10^8$ |
| # multiplications | | | |
| Standard processing | $4.9 \cdot 10^8$ | $9.6 \cdot 10^8$ | $1.9 \cdot 10^9$ |
| Batch processing | $7.4 \cdot 10^7$ | $7.4 \cdot 10^7$ | $7.4 \cdot 10^7$ |

**Table 6.9:** Comparison of the amount of additions and multiplications per second involved in the standard and the batch processing (without carrier and code generation).

The following observations can be concluded from the tables above:

- the complexity of the batch processing is reduced by roughly one order of magnitude with respect to the standard processing, when considering a sampling frequency of 8 MHz or higher;
- the amount of multiplications is now independent of the sampling frequency;
- the complexity is split into the unavoidable basic load of the chip sums computation (common to all the satellite channels) and the additional tasks inherent to each satellite channel;
- the impact of the signals quantization (incoming satellite signal and local carrier replica) on the receiver complexity is quasi negligible, since all the operations are performed with integer arithmetic and do not depend on the operands bit-depth. This makes the architecture flexible and open to various signal configurations;
- the carrier is real-time synthesized with a complexity proportional to the residual Doppler frequency. The waveform is generated continuously at any desired frequency, which makes it possible to compensate natively for any residual Doppler;
- the code is real-time synthesized with a complexity proportional to the code chipping rate. The waveform is generated continuously at any desired frequency, which makes it possible to compensate natively for any residual Doppler;
- the memory requirements are reduced to the strict minimum, as no oversampled carrier or code sequences need to be stored.

In conclusion, the batch processing drastically reduces the complexity of the base-band architecture, leading to an amount of operations which is now fully suitable for a real-time implementation.

## 6.2 Architecture trade-offs

### 6.2.1 Effects of the constant chip sums size

Since the size and the samples boundaries of the chips depend on the code frequency, a chip sums sequence theoretically needs to be computed for each of the E, P, and L signal components, in parallel for all the satellite channels. This results in a very large amount of operations ($\propto 6 \cdot f_s \cdot N_c$), not suitable for a real-time application. In order to simplify the chip sums computation, their summation interval is set constant and of length $\overline{B}$. This assumption bears no consequences as long as the constant chip sum length equals the initial one (i.e. $\overline{B} = B^x_{k+1} - B^x_k$), but introduces an error when it differs. The effect of accumulating the wrong number of samples in the $k^{th}$ chip sum can occur in two different ways, depending on the sampling frequency and thus, on the value of $\overline{B}$:

1. if $\overline{B} = \text{ceil}(\overline{b})$, the sample $B^x_k + \overline{B} - 1$ is integrated twice, in two consecutive chip sums;
2. if $\overline{B} = \text{floor}(\overline{b})$, the sample $B^x_k + \overline{B}$ is dropped.

The impact on the correlation process depends on the probability $P_e$ of having a sample erroneously integrated or dropped that can be estimated as:

$$P_e = \frac{|\overline{b} - \overline{B}|}{\overline{b}} \tag{6.6}$$

Consequently, the smaller the difference $|\overline{b} - \overline{B}|$ is, the lesser the impact on the autocorrelation function is, the worst case occurring when $|\overline{b} - \overline{B}| = 0.5$. For example, with a sampling frequency $f_s = 8$ MHz and accordingly to Equation 5.8, $\overline{b} = 7.82$ and the chip sums size is set to $\overline{B} = 8$ samples. This corresponds to the first case described above (i.e. $\overline{B} = \text{ceil}(\overline{b})$), and thus, according to Equation 6.6, the probability of integrating the same value twice is $P_e = 0.023$ (approximately once every $43^{th}$ sample). This example has been simulated and is illustrated in Figure 6.1.

The right part of the peak remains unaffected while the left one is slightly shifted, proportionally to the number of erroneously integrated samples ($N_s \cdot P_e = 8000 \cdot 0.023 = 184$ samples for the illustrated example). However, since the probability of having a corrupted sample remains quasi constant for all the satellite channels, this translates into a common correlation peak offset that finally does not impact the PVT solution computation. Consequently, with an appropriate selection of the sampling frequency, the effect of using a constant chip sums size is quasi negligible.

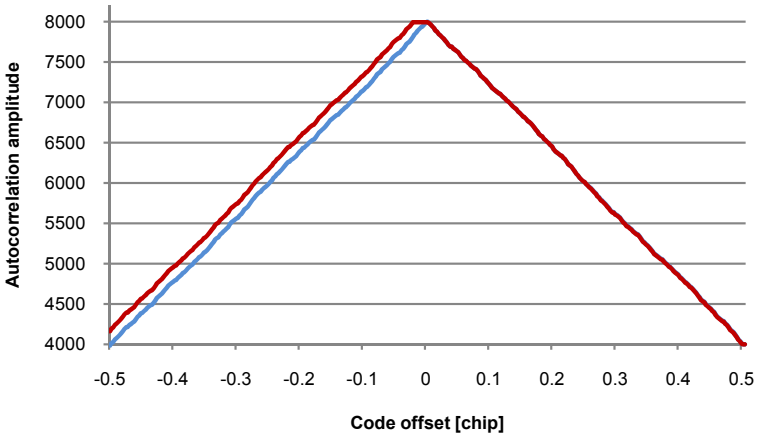


**Figure 6.1:** Effect of using a constant chip sums size on the autocorrelation function with a sampling frequency $f_s = 8$ MHz and an integration time $T_{int} = 1$ ms. The blue curve represents the original function, the red curve the affected one.

### 6.2.2 Effects of the carrier boundaries approximation

The partial sums are obtained by accumulating consecutive chip sums accordingly to the sequence $R_j^x$ defined in Equation 5.16. Since the number of chips per carrier interval most likely results in a non-integer value, the chip sums contained in each $j^{th}$ carrier interval are those which overlap, either completely or by more than 50% at the boundaries of the interval. Consequently, for each interval transition, the maximal number of samples erroneously introduced in a partial sum is limited to half a chip. This represents a few samples as compared to the partial sums size of several hundreds or thousands samples. In the worst case, the percentage of samples erroneously integrated in one carrier interval can be estimated as:

$$\frac{\overline{a}}{\overline{\overline{b}}} = \frac{f_d \cdot 2^{M+1}}{2 \cdot f_c} \tag{6.7}$$

Assuming a 3-bit carrier quantization, this represents less than 4% of the samples contained in one carrier interval. Furthermore, since the effect is randomly distributed over the different intervals and averaged over one integration period, it bears no consequence on the carrier NCO.

### 6.2.3 Code replica time delay

The time delay between the E, P, and L code replica is emulated by shifting the boundaries of the chip sums by a fixed number of samples $\Delta n$. This restricts the available shifts to integer multiples of the sampling period $T_s$ (i.e. the minimal spacing between the E and P or P and L replicas is one sample). Consequently, the higher the sampling frequency is, the thinner the replica spacing can be. However, this limitation can be resolved by running three distinct samples accumulators independently, each with the desired offset $b_{off}$.

## 6.3 Extension to Galileo E1 OS

As compared to the BPSK modulation of the GPS L1 CA, the Galileo E1 OS signals requires the additional sub-carrier square wave to be removed during the code wipe-off process. The discussion focuses now on the BOC(1,1) modulation since it holds more than 90% of the signal energy (cf. Equation 2.11). The effect of the BOC(1,1) on the code is to split each chip into two sign-inverted halves, as illustrated in Figure 6.2.

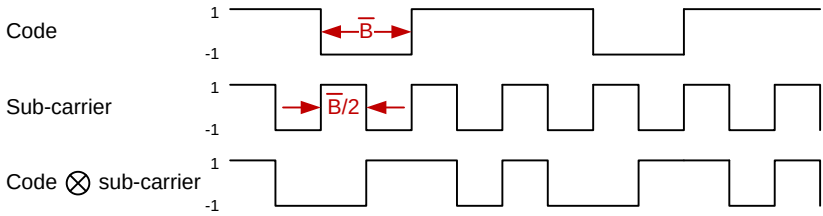


**Figure 6.2:** Square wave sub-carrier modulation of the BOC(1,1).

With an appropriate selection of the sampling frequency, the sub-carrier removal can be directly integrated within the chip sums computation. If the number of samples per chip rounded to the nearest integer $\overline{B}$ actually results in an even value, the chip sums summation interval can be split into two halves of $\frac{\overline{B}}{2}$ samples. The requirement for the sampling frequency can be stated as:

$$\text{round}\left(\frac{f_s}{f_c}\right) = 2 \cdot \text{round}\left(\frac{f_s}{2 \cdot f_c}\right) \tag{6.8}$$

If the above criterion is fulfilled, each chip sum $S_n$ can now be formed by summing up $\frac{\overline{B}}{2}$ consecutive data values and subtracting the next $\frac{\overline{B}}{2}$ consecutive ones, starting from the sample $n$ in question, as follows:

$$S_n = \sum_{m=n}^{n+\frac{\overline{B}}{2}-1} (I(m) + i \cdot Q(m)) - \sum_{m=n+\frac{\overline{B}}{2}}^{n+\overline{B}-1} (I(m) + i \cdot Q(m)) \tag{6.9}$$

The computation of the chip sums can be performed iteratively with four additions and four subtractions, as follows:

$$S_0 = \sum_{n=0}^{\frac{\overline{B}}{2}-1} (I(n) + i \cdot Q(n)) - \sum_{n=\frac{\overline{B}}{2}}^{\overline{B}-1} (I(n) + i \cdot Q(n)) \tag{6.10}$$

$$\begin{aligned} S_{n+1} = S_n &+ 2 \cdot I(n + \frac{\overline{B}}{2}) - I(n + \overline{B}) - I(n) \\ &+ i \cdot [2 \cdot Q(n + \frac{\overline{B}}{2}) - Q(n + \overline{B}) - Q(n)] \qquad 0 \leq n < N_s - 1 \end{aligned}$$

The chips sums are chronologically collected into a vector. The latter is addressed by means of the code accumulator $b_k$ and further processed as described in Subsection 5.3.3.

In addition to the unavoidable load and store operations, Table 6.10 provides a rough estimation of the amount of integer operations per second necessary to process $N_c$ satellite channels with the new batch processing based architecture (without considering the carrier and code generation). We assume here three different code replicas, namely E, P, and L.

| | # additions | # multiplications |
|---|---|---|
| Chip sums | $8 \cdot f_s$ | - |
| Code mixing | - | $6 \cdot f_c \cdot N_c$ |
| Carrier mixing | $6 \cdot 2^{M+1} \cdot N_c / T_{int}$ | $12 \cdot 2^{M+1} \cdot N_c / T_{int}$ |
| Accumulation | $6 \cdot f_c \cdot N_c$ | - |

**Table 6.10:** Amount of integer operations per second for the processing of the BOC(1,1) modulation (without code and carrier generation).

As compared to Table 6.8, the BOC(1,1) modulation only requires twice more iterations for the chip sums computation, which has no significant impact on the overall complexity. With an appropriate selection of the sampling frequency, the effect of the additional sub-carrier modulation can thus be restricted to the chip sums computation, without penalizing the other operations.

However, the GPS L2 P(Y) as well as some of the modernized signals modulations (e.g. Galileo E5-A and E5-B [Eur10a]) rely on higher chipping rates, and consequently higher sampling frequencies. If the effect of the latter is limited to the chip sums computation, the impact of the code frequency on the complexity is preponderant, as their dependence is quasi linear. For example, increasing the code frequency from 1.023 to 10.23 MHz affects the overall complexity by nearly a factor of ten. Nevertheless, the batch processing still remains extremely efficient, as compared to the integer implementation whose complexity is proportional to the sampling rate (see Table 3.2) and thus penalized in the same proportions by the frequency increase.

## 6.4 Summary

The key point of the proposed solution relies on the progressive data throughput reduction, which is first decreased from the sampling rate of several megahertz to the code rate of one megahertz, and finally to a frequency proportional to the number of carrier phases. As compared to a traditional hardware based receiver, where all the operations are performed at the sampling frequency, the amount of integer additions and multiplications involved in the base-band processing is reduced by almost one order of magnitude. Furthermore, performing all the operations with integer arithmetic allows the signals structure to change significantly without any code modification. This architecture thus provides great flexibility and can easily accommodate various receiver configurations (signal and carrier quantization, sampling frequency, etc.).

# Chapter 7

# Implementation of the new architecture

The proposed receiver architecture was implemented in a demonstrator for validating the different algorithms and demonstrating the feasibility of the concept. It was successfully tested with different signal sources (simulated and real ones), on various computers equipped with different microprocessors. The details and the performance of the implementation are presented in this chapter.

## 7.1 Demonstrator description

The demonstrator consists in a RF front-end receiving the satellites signals either from an external antenna or a GPS simulator. The captured signal is properly conditioned, digitized, and further transmitted to the host computer via the USB 2.0 interface. The software receiver processes the incoming samples in order to extract the pseudoranges. These are transmitted to the navigation solution that computes the PVT solution and forwards the results to be displayed. The system is depicted in Figure 7.1.

An external aiding receiver (*u-blox 5* GPS receiver [U-b10]) is connected to the same signal source as the software receiver. The aiding receiver acquires all the visible satellites, decodes the ephemeris data and the time information (GPS week). These parameters are polled by the software receiver - to determine the list of satellites to search for - and forwarded to the PVT solution (provided by *u-blox*).

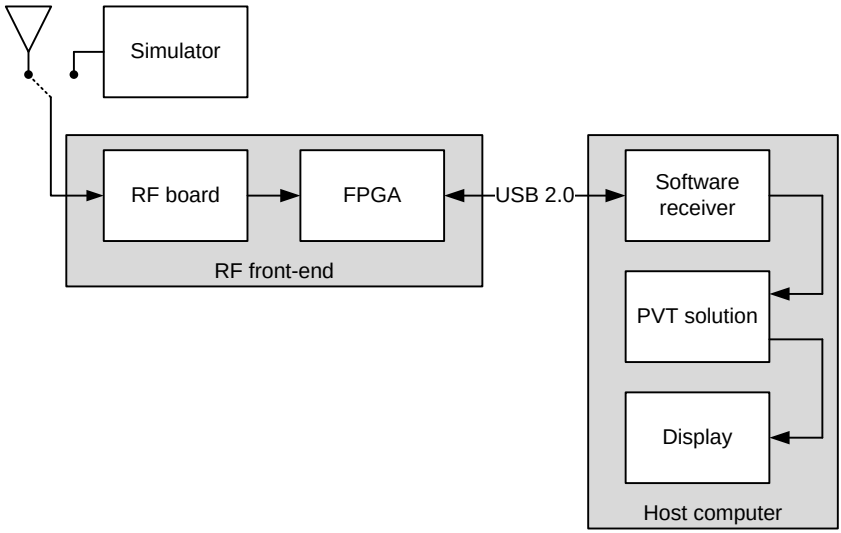


**Figure 7.1:** Schema bloc of the demonstrator. The receiver is composed of an external RF front-end unit connected to the host computer via USB 2.0. The aiding receiver is not shown.

The different components of the demonstrator are detailed in the next sections.

### 7.1.1 RF front-end

We make hereafter the distinction between the RF front-end, which consists in the whole hardware part preceding the host computer, and the RF board which is a single component of the latter. The RF front-end is composed of two interconnected parts, namely the RF board and the FPGA board.

#### RF board

The RF board is provided by the industrial partner *u-blox AG*. It consists in a low-IF architecture with a quadrature down-conversion, completed by a low noise amplifier at the input, several filters, and additional gain stages. The board also embeds a dual channel 8-bit ADC providing a complex digital output stream. A picture is shown in Figure 7.2.

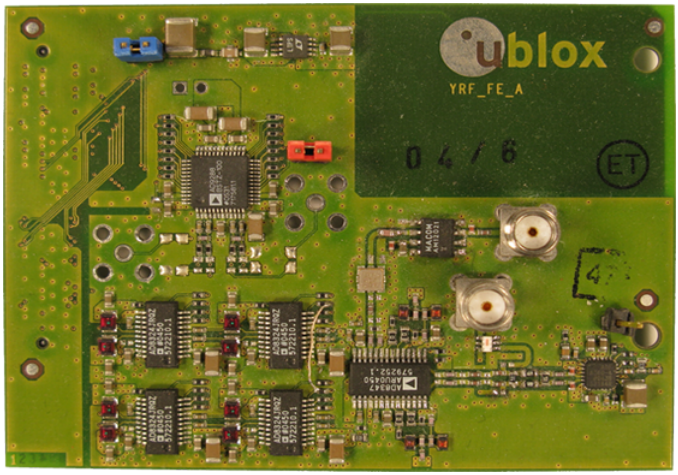


**Figure 7.2:** RF board (provided by *u-blox AG*) with a dual-channel ADC.

The on board components are driven by an external 24 MHz reference clock that fixes the data sampling frequency to 24 MHz and the carrier IF to 3.42 MHz. The RF board is completed by an external FPGA board, interfacing with the receiver and hosting additional digital processing.

### FPGA board

The FPGA board [FPG10] interfaces the RF board to the host computer. It relies on an *Altera Cyclone II* FPGA chip [Alt08] and also embeds a *Cypress* USB 2.0 controller [Cyp06] for ensuring high speed data transfer with the host. Furthermore, the board also provides the 24 MHz reference clock to the RF board via a dedicated interconnecting Printed Circuit Board (PCB). Additional digital base-band pre-processing is implemented in the FPGA, mainly consisting in a mixer, for down-converting the carrier IF to base-band, and a data bandwidth reduction stage for lowering the samples bit rate (see Section 7.2 for more details). A picture of the board is shown in Figure 7.3.

## 7.1.2 Host computer

The host running the software receiver is a standard Personal Computer (PC) installed with *Windows XP Professional SP3* operating system.

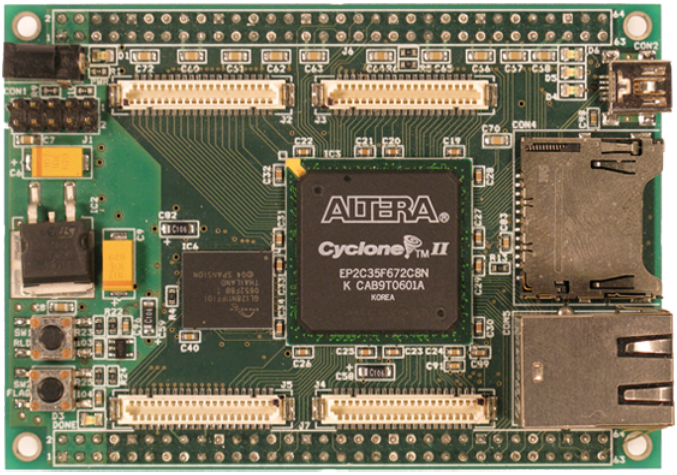


**Figure 7.3:** FPGA board with embedded USB 2.0 controller.

## 7.2 Base-band pre-processing implementation

The base-band pre-processing is a top level unit common to all the satellite channels and that is implemented in Very high speed integrated circuit High Density Language (VHDL) in the FPGA. It is responsible for three main functionalities:

1. the carrier IF removal;
2. the data filtering;
3. the data bandwidth reduction.

A block diagram of the base-band pre-processing stage is illustrated in Figure 7.4. The RF board provides a complex digital stream sampled at 24 MHz with 8-bit data resolution. The signal is first high-pass filtered to remove any residual Direct Current (DC) component before being down-converted to baseband within a mixer operating at 3.42 MHz. The latter frequency can be adjusted in order to compensate for the quartz offset. The data resolution is also decreased to 3 bits during the mixing operations. The base-band signal is then lowpass filtered and the sampling frequency is decimated from 24 MHz to 8 MHz, the nominal frequency of the receiver operations.

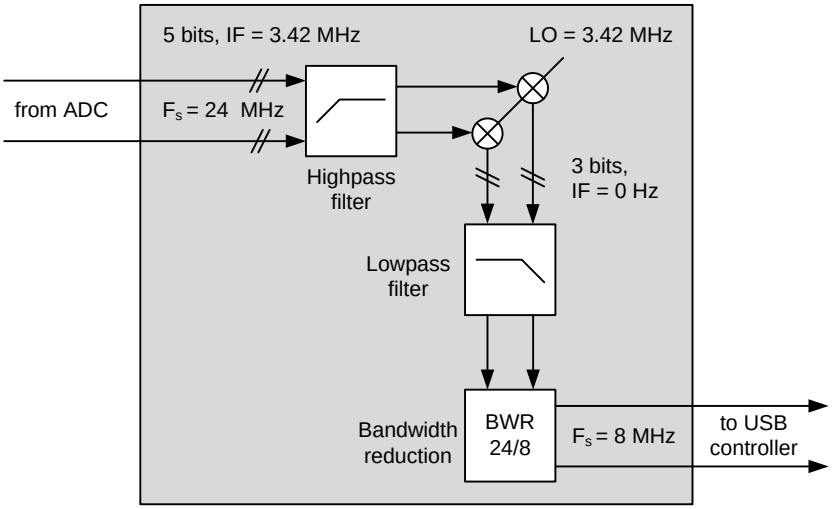


**Figure 7.4:** Base-band pre-processing stage with respectively high-pass filtering, carrier down-conversion, lowpass filtering and data bandwidth reduction.

The characteristics of the signal transmitted to the receiver are regrouped in Table 7.1.

| | Signal properties |
|---|---|
| Data resolution | 3-bit I & Q |
| Data sampling frequency | 8 MHz |
| Carrier IF | 0 Hz |
| Residual Doppler range | ±5 kHz |

**Table 7.1:** Characteristics of the receiver incoming signal I(n) and Q(n).

The signal samples $I(n)$ and $Q(n)$ are originally represented as 3-bit signed integers in the range [-4; 3]. However, to simplify their transmission to the host computer and their future manipulation within the receiver, they are converted into 3-bit unsigned integers in the range [0; 7] and rearranged to form a 8-bit data word $D(n)$. The structure of each data word is represented in Table 7.2.

| Data word D(n) | | | | | | | |
|---|---|---|---|---|---|---|---|
| $D_7(n)$ | $D_6(n)$ | $D_5(n)$ | $D_4(n)$ | $D_3(n)$ | $D_2(n)$ | $D_1(n)$ | $D_0(n)$ |
| 0 | $I_2(n)$ | $I_1(n)$ | $I_0(n)$ | 0 | $Q_2(n)$ | $Q_1(n)$ | $Q_0(n)$ |

**Table 7.2:** 8-bit data organization for the USB transfer. $I(n)$ and $Q(n)$ are encoded as 3-bit unsigned integer.

Sampling the signal at the frequency of 8 MHz represents a data rate of 8 MB/s to be handled by the USB 2.0 interface. The continuous data streaming is essential to keep the synchronization between the receiver and the respective satellite signals. For this reason, the USB 2.0 controller is configured in *isochronous* mode which provides time-critical data delivery, like in audio and video applications, and guarantees the appropriate data bandwidth during the transfers [Cyp06].

To ensure the uninterrupted handling of the incoming data, the samples are alternately stored into two buffers of 160.000 bytes (corresponding to a 20 ms data sequence) within the host computer. While the USB controller fills one of the two buffers with the new samples, the receiver processes the previous 20 ms data sequence stored in the other one, and vice versa. The principle is illustrated in Figure 7.5.

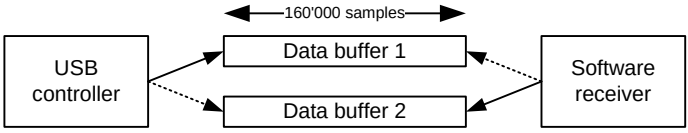


**Figure 7.5:** Twin data buffer. Each buffer is alternately accessed by the USB controller and the receiver.

## 7.3 Base-band processing implementation

All the base-band processing operations are implemented in C programming language and compiled with *Microsoft Visual Studio 2008*.

We assume from now that the complex signal entering the receiver has the properties regrouped in Table 7.1. We also assume a basic coherent integration time $T_{int} = 0.001$ s, which fixes the number of samples per integration to $N_s = 8000$. From each received data byte $D(n)$, the samples $I(n)$ and $Q(n)$, encoded as 3-bit unsigned integers, can be easily retrieved with the following operations:

$$\begin{aligned} I(n) &= D(n) \gg 4 \\ Q(n) &= D(n) \ \ \& \ \ 7 \end{aligned} \tag{7.1}$$

where $\gg$ represents the logical right shift operator;
$\&$ represents the logical AND operator.

Note that the reconversion from the unsigned to the signed integer representation of the data occurs during the chip sums computation.

For each sample $I(n)$ and $Q(n)$, a complex partial sum is formed by accumulating consecutive samples appertaining to the same chip. Considering the nominal code frequency, the average number of samples $\overline{b}$ per chip is given by Equation 5.8 as:

$$\overline{b} = \frac{f_s}{f_c} = \frac{8 \cdot 10^6}{1.023 \cdot 10^6} = 7.82 \text{ samples} \tag{7.2}$$

Practically, depending on the sampling phase and the code offset, the number of samples per chip is either seven or eight (except for the first and last chip which may be incomplete). However, in order to simplify the chip sums computation, the summation interval is assumed constant and, accordingly to Equation 5.20, becomes:

$$\overline{B} = \text{round}(\overline{b}) = 8 \text{ samples} \tag{7.3}$$

Every chip sum is now composed of the sum of eight consecutive samples. The assumption of having a constant chip sums size introduces an error in all the chip sums originally made of seven samples. Accordingly to Equation 6.6, the probability $P_e$ of having a sample which is integrated twice is:

$$P_e = \frac{\left|\overline{b} - \overline{B}\right|}{\overline{b}} = \frac{|7.82 - 8|}{7.82} = 0.023 \tag{7.4}$$

This implies integrating approximately every $43^{th}$ sample twice and thus corrupts roughly 184 samples per millisecond (see Figure 6.1).

With the summation interval constant, the 8000 chip sums can now be computed recursively with only four operations, as shown in Equation 7.5. This is to be compared to the original chip sum computation that originally required 14 additions.

$$S_0 = \sum_{n=0}^{7}(I(n) - 4 + i \cdot Q(n) - i \cdot 4) \tag{7.5}$$

$$S_{n+1} = S_n + I(n+8) - I(n) + i \cdot [Q(n+8) - Q(n)] \quad 0 \leq n < 7'999$$

The unsigned to signed integer reconversion of the samples is performed in the first partial sum $S_0$, by subtracting the value 4, and propagates iteratively. The so computed chip sums are chronologically regrouped in a vector that takes the form of Table 7.3.

| | | Complex chip sums |
|---|---|---|
| Sample index $n$ | 0 | $S_0 = S(0{:}7)$ |
| | 1 | $S_1 = S(1{:}8)$ |
| | 2 | $S_2 = S(2{:}9)$ |
| | ⋮ | ⋮ |
| | 7'999 | $S_{7'999} = S(7'999{:}8'006)$ |
| | ⋮ | 0 |

**Table 7.3:** Complex chip sums vector with a constant chip sums size of 8 samples.

The vector is now addressed by means of the code accumulator providing the samples boundaries of each chip through Equation 5.9. Assuming a zero initial code offset (i.e. $b_{off} = 0$), the accumulator $b_k$ is incremented by the average number of samples per chip $\overline{b} = 7.82$ as follows:

$$b_k = k \cdot 7.82 \qquad 0 \leq k < K + 1 = 1024 \tag{7.6}$$

Table 7.4 provides some values of the code accumulator as example.

| | Chip index $k$ | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | ... | 1023 |
| $b_k$ | 0 | 7.82 | 15.64 | 23.46 | ... | 8000 |
| $B_k$ | 0 | 7 | 15 | 23 | ... | 8000 |
| $\Delta B_k$ | 7 | 8 | 8 | 8 | ... | 8 |

**Table 7.4:** Code accumulator $b_k$ incremented by the average number of samples per chip $\bar{b} = 7.82$, samples boundaries $B_k$, and number of samples per chip $\Delta B_k = B_{k+1} - B_k$.

In order to emulate the time shift between the code replicas, the chip sums boundaries $B_k$ are shifted by the desired amount of samples. We assume here three samples replica spacing (i.e. $\Delta n = 3$), which corresponds to less than half a chip spacing. The indexes $B_k^E$, $B_k^P$, and $B_k^L$ are computed accordingly to Equation 5.11. Table 7.5 provides some values of the accumulator as example.

| | Chip index $k$ | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | ... | 1023 |
| $B_k^E$ | 0 | 7 | 15 | 23 | ... | 8000 |
| $B_k^P$ | 3 | 10 | 18 | 26 | ... | 8003 |
| $B_k^L$ | 6 | 13 | 21 | 29 | ... | 8006 |

**Table 7.5:** Chip sums boundaries $B_k^E$, $B_k^P$, and $B_k^L$ for the E, P, and L code replicas respectively.

For each satellite channel, the chip sums vector is addressed by means of $B_k^E$, $B_k^P$, and $B_k^L$ for providing the appropriate E, P, and L chip sums sequences as follows:

$$\begin{aligned} &\{S_0, \quad S_7, \quad S_{15}, \quad \ldots \quad S_{8000}\} \\ &\{S_3, \quad S_{10}, \quad S_{18}, \quad \ldots \quad S_{8003}\} \\ &\{S_6, \quad S_{13}, \quad S_{21}, \quad \ldots \quad S_{8006}\} \end{aligned} \tag{7.7}$$

Each sequence is now multiplied point by point with the PRN code to perform the code wipe-off. The products are then distributed into the different carrier intervals in order to form the partial sums. Assuming a local carrier resolution of 3 bits defining 16 different phases, the average size of the partial sums $\overline{a}$ is given by Equation 5.2. Considering the highest Doppler of 5 kHz, this leads to an average number of samples per carrier interval of:

$$\overline{a} = \frac{f_s}{2^{M+1} \cdot f_d} = \frac{8 \cdot 10^6}{16 \cdot 5'000} = 100 \text{ samples} \tag{7.8}$$

The average number of chip per carrier interval can now be estimated accordingly to Equation 5.14 as:

$$\overline{r} = \frac{\overline{a}}{\overline{b}} = \frac{100}{7.82} = 12.79 \text{ chips} \tag{7.9}$$

The carrier interval boundaries are provided by the carrier accumulator $r_j$, incremented by the average number of chips per carrier interval $\overline{r} = 12.79$, as follows:

$$\begin{aligned} r_j &= j \cdot 12.79 \qquad & 0 \leq j < 80 \\ r_{80} &= 1023 \end{aligned} \tag{7.10}$$

Table 7.6 provides some values of the accumulator as well as the chips boundaries of the carrier intervals computed accordingly to Equation 5.16.

| | Carrier interval $j$ | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | ... | 80 |
| $r_j$ | 0 | 12.79 | 25.58 | 38.36 | ... | 1023 |
| $R_j^E$ | 0 | 13 | 26 | 38 | ... | 1023 |
| $R_j^P$ | 0 | 12 | 25 | 38 | ... | 1023 |
| $R_j^L$ | 0 | 12 | 25 | 38 | ... | 1022 |

**Table 7.6:** Carrier accumulator $r_j$ incremented by the average number of chips per carrier interval $\overline{r} = 12.79$ and interval boundaries $R_j^x$.

The partial correlations are now distributed and integrated over the different carrier intervals accordingly to $R_j^E$, $R_j^P$, and $R_j^L$. This results in three E, P, and L sequences of partial sums $P_j^x$ regrouped in Table 7.7.

| | | Complex partial sums $P_j^x$ | | |
|---|---|---|---|---|
| Carrier interval $j$ | 0 | $P^E$(0:12) | $P^P$(0:11) | $P^L$(0:11) |
| | 1 | $P^E$(13:25) | $P^P$(12:24) | $P^L$(12:24) |
| | 2 | $P^E$(26:37) | $P^P$(26:38) | $P^L$(26:38) |
| | ⋮ | ⋮ | ⋮ | ⋮ |
| | 79 | $P^E$(1010:1022) | $P^P$(1010:1022) | $P^L$(1010:1021) |

**Table 7.7:** Complex partial sums associated to each carrier interval $j$.

Thanks to the carrier cyclicity, all the $16^{th}$ partial sums are regrouped and summed up to be multiplied only once by the same carrier magnitude. This way, the partial sums are reduced to 16 batches $T_l^x$, each one associated to a carrier phase $l$=mod($j$, 16). The matrix of Table 7.7 takes the form of Table 7.8.

| | | Reduced complex partial sums $T_l^x$ | | |
|---|---|---|---|---|
| Carrier phase $l$ | 0 | $P_0^E$+$P_{16}^E$ ... +$P_{64}^E$ | $P_0^P$+$P_{16}^P$ ... +$P_{64}^P$ | $P_0^L$+$P_{16}^L$ ... +$P_{64}^L$ |
| | 1 | $P_1^E$+$P_{17}^E$ ... +$P_{65}^E$ | $P_1^P$+$P_{17}^P$ ... +$P_{65}^P$ | $P_1^L$+$P_{17}^L$ ... +$P_{65}^L$ |
| | 2 | $P_2^E$+$P_{18}^E$ ... +$P_{66}^E$ | $P_2^P$+$P_{18}^P$ ... +$P_{66}^P$ | $P_2^L$+$P_{18}^L$ ... +$P_{66}^L$ |
| | ⋮ | ⋮ | ⋮ | ⋮ |
| | 15 | $P_{15}^E$+$P_{31}^E$ ... +$P_{79}^E$ | $P_{15}^P$+$P_{31}^P$ ... +$P_{79}^P$ | $P_{15}^L$+$P_{31}^L$ ... +$P_{79}^L$ |

**Table 7.8:** Reduced complex partial sums associated to each carrier phase $l$.

The carrier removal is now operated by multiplying each of the 16 reduced complex partial sums $T_l^x$ of Table 7.8 by the respective carrier magnitudes $\{\pm 1, \pm 3, \pm 4, \pm 5\}$ given in Table 2.3.

The products are then rearranged accordingly to Equation 2.20 to form the definitive correlation results $I^x$ and $Q^x$ as follows:

$$\begin{aligned}
I^x = &[\mathrm{Re}(T_4^x) + \mathrm{Re}(T_{11}^x) - \mathrm{Re}(T_3^x) - \mathrm{Re}(T_{12}^x)] \\
&+ [\mathrm{Re}(T_5^x) + \mathrm{Re}(T_{10}^x) - \mathrm{Re}(T_2^x) - \mathrm{Re}(T_{13}^x)] \cdot 3 \\
&+ [\mathrm{Re}(T_6^x) + \mathrm{Re}(T_9^x) - \mathrm{Re}(T_1^x) - \mathrm{Re}(T_{14}^x)] \cdot 4 \\
&+ [\mathrm{Re}(T_7^x) + \mathrm{Re}(T_8^x) - \mathrm{Re}(T_0^x) - \mathrm{Re}(T_{15}^x)] \cdot 5 \\
&+ [\mathrm{Im}(T_0^x) + \mathrm{Im}(T_7^x) - \mathrm{Im}(T_8^x) - \mathrm{Im}(T_{15}^x)] \\
&+ [\mathrm{Im}(T_1^x) + \mathrm{Im}(T_6^x) - \mathrm{Im}(T_9^x) - \mathrm{Im}(T_{14}^x)] \cdot 3 \\
&+ [\mathrm{Im}(T_2^x) + \mathrm{Im}(T_5^x) - \mathrm{Im}(T_{10}^x) - \mathrm{Im}(T_{13}^x)] \cdot 4 \\
&+ [\mathrm{Im}(T_3^x) + \mathrm{Im}(T_4^x) - \mathrm{Im}(T_{11}^x) - \mathrm{Im}(T_{12}^x)] \cdot 5
\end{aligned}$$

$$\begin{aligned}
Q^x = &[\mathrm{Im}(T_4^x) + \mathrm{Im}(T_{11}^x) - \mathrm{Im}(T_3^x) - \mathrm{Im}(T_{12}^x)] \\
&+ [\mathrm{Im}(T_5^x) + \mathrm{Im}(T_{10}^x) - \mathrm{Im}(T_2^x) - \mathrm{Im}(T_{13}^x)] \cdot 3 \\
&+ [\mathrm{Im}(T_6^x) + \mathrm{Im}(T_9^x) - \mathrm{Im}(T_1^x) - \mathrm{Im}(T_{14}^x)] \cdot 4 \\
&+ [\mathrm{Im}(T_7^x) + \mathrm{Im}(T_8^x) - \mathrm{Im}(T_0^x) - \mathrm{Im}(T_{15}^x)] \cdot 5 \\
&- [\mathrm{Re}(T_0^x) + \mathrm{Re}(T_7^x) - \mathrm{Re}(T_8^x) - \mathrm{Re}(T_{15}^x)] \\
&- [\mathrm{Re}(T_1^x) + \mathrm{Re}(T_6^x) - \mathrm{Re}(T_9^x) - \mathrm{Re}(T_{14}^x)] \cdot 3 \\
&- [\mathrm{Re}(T_2^x) + \mathrm{Re}(T_5^x) - \mathrm{Re}(T_{10}^x) - \mathrm{Re}(T_{13}^x)] \cdot 4 \\
&- [\mathrm{Re}(T_3^x) + \mathrm{Re}(T_4^x) - \mathrm{Re}(T_{11}^x) - \mathrm{Re}(T_{12}^x)] \cdot 5
\end{aligned} \tag{7.11}$$

## 7.4 Base-band algorithms implementation

All the base-band processing operations are implemented in C programming language and compiled with *Microsoft Visual Studio 2008*. The FFTs are computed by means of the open-source *fftw3* library [Fri07].

### 7.4.1 Acquisition algorithms implementation

The receiver should accommodate a large Doppler range, mainly due to the quartz offset which can be as large as ±40 kHz. Consequently, the acquisition takes place in two steps, by first searching for all the satellites over the full Doppler uncertainty with the parallel frequency search. Based on this information, all satellites are re-acquired on specific frequency bins (since the code phase estimation may be out of date due to a possible quartz drift) by means of the parallel code search.

### Parallel frequency search implementation

The first acquisition is implemented as parallel frequency search, characterized by the parameters summarized in Table 7.9.

| Configuration of the parallel frequency search | |
|---|---|
| Total coherent integration time $T_{int} = N_p \cdot T_p$ | 10 ms |
| Pre-detection time $T_p$ | 9.76 $\mu$s |
| FFT size $N_p$ | 1024 points |
| Frequency search bandwidth $\frac{1}{T_p}$ | 102'400 Hz |
| Frequency bin width $\frac{1}{N_p \cdot T_p}$ | 100 Hz |
| Code step size $\Delta n$ | 1 sample |

**Table 7.9:** Characteristics of the incoming signal.

This configuration leads to the power envelope illustrated in Figure 7.6.

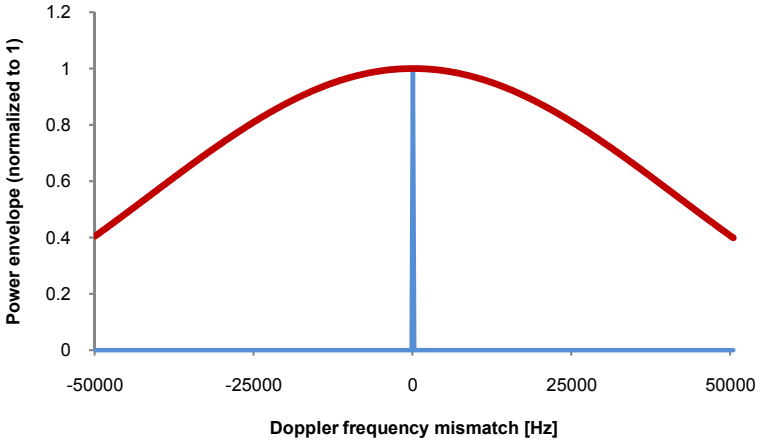


**Figure 7.6:** Global power envelope of the parallel frequency search.

A sequence of 10'230 chip sums is selected from the chip sums vector, accordingly to (cf. Equation 5.27):

$$\{S_0, \quad S_7, \quad S_{15}, \quad \ldots \quad S_{79992}\} \tag{7.12}$$

The code sequence of 10'230 chips is obtained by repeating 10 times the original PRN code. Both sequences are multiplied point by point to remove the code and the products are distributed over the 1024 pre-detection sums. The average number of chip sums per pre-detection sum is given by Equation 5.30:

$$\overline{v} = T_p \cdot f_c = 9.76 \cdot 10^{-6} \cdot 1.023 \cdot 10^6 = 9.99 \text{ chip sums} \tag{7.13}$$

The chips boundaries of the pre-detection sums are computed through Equation 7.14 by accumulating $\overline{v}$:

$$v_i = i \cdot 9.99 \qquad 0 \leq i < 1024 \tag{7.14}$$

Some values of the accumulator are given in Table 7.10 as example.

| | Pre-detection sum $i$ | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | ... | 1023 |
| $v_i$ | 0 | 9.99 | 19.98 | 29.97 | ... | 10220 |
| $V_i$ | 0 | 10 | 20 | 30 | ... | 10220 |

**Table 7.10:** Accumulator $v_i$ incremented by the average number of chips sums per pre-detection sum $\overline{v} = 9.99$, and pre-detection sums boundaries $V_i$.

The pre-detection sums sequence is formed by summing up the partial correlations, accordingly to $V_i$, and transformed into the frequency domain via FFT. The energy of each data bin is computed and compared to the threshold to declare if the satellite is present or not. In order to recover the loss that occurs when the Doppler shift falls between the two bins, the difference of two adjacent bins response is also computed. This way, it is theoretically possible to recreate a frequency component at the center between the bins and thus recover about 3 dB of sensitivity [Tsu05]. This is illustrated in Figure 7.7.

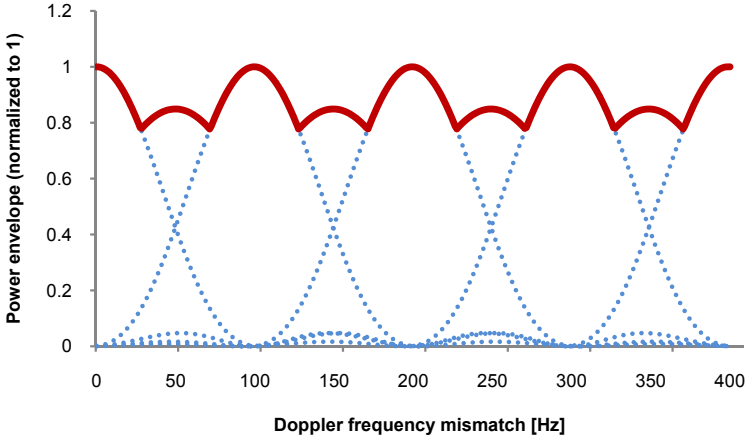


**Figure 7.7:** The blue curve represents the original frequency bins response with 100 Hz bin spacing. The red curve is obtained by subtracting and normalizing two adjacent bins response.

In order to test the 1023 code phases with one eighth of chip step size, eight chip sums sequences are selected with the summation boundaries shifted by one sample with respect to each other:

$$\begin{array}{lllll} \{S_0, & S_7, & S_{15}, & \dots & S_{79992}\} \\ \{S_1, & S_8, & S_{16}, & \dots & S_{79993}\} \\ \{S_2, & S_9, & S_{17}, & \dots & S_{79994}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \{S_7, & S_{14}, & S_{22}, & \dots & S_{79999}\} \end{array} \tag{7.15}$$

For each sequence, the whole search procedure is repeated by circular-shifting 1022 times the code replica vector. This makes a maximum of $8 \cdot 1023 = 8184$ different code phases to test. To get rid of a possible data bit transition, the whole search is performed on two consecutive data sets of 10 ms.

### Parallel code search implementation

The parallel code search operates on specific frequency bins, accordingly to the information provided by the first acquisition stage. Table 7.11 summarizes the re-acquisition parameters.

| Configuration of the parallel code search | |
|---|---|
| Total coherent integration time $T_{int}$ | 7 ms |
| FFT size $N$ | 8192 points |
| Code step size $\Delta n$ | 1 sample |

**Table 7.11:** Configuration of the parallel frequency search.

A sequence of 8'184 chip sums is selected from the chip sums vector, accordingly to (cf. Equation 5.24):

$$\{S_0, \quad S_7, \quad S_{15}, \quad \dots \quad S_{62553}\} \tag{7.16}$$

The 8'184 points local carrier is generated accordingly to the interval boundaries $R_j^E$ in Equation 5.15, and the respective carrier magnitudes given in Table 2.3. Considering a 5 kHz frequency, this gives:

$$\{5-i, \quad 5-i, \quad 5-i, \quad \dots \quad 5+i\} \tag{7.17}$$

Both sequences are multiplied point by point and the result is left zero padded (eight additional values are added) in order to form a vector of 8'192 points, accordingly to Equation 5.26. The latter is then multiplied point by point in the frequency domain with the FFT of seven consecutive PRN code periods (right zero padded with 1031 values to discard the end effect), in order to form a vector of 8'192 points on which the inverse FFT is executed. The energy of each data point is computed and compared to the threshold to declare if the satellite is present or not.

In order to test the 1023 code phases with one eighth of chip step size, eight chip sums sequences are selected, with the summation boundaries shifted by one sample with respect to each other, as in Equation 7.15. To get rid of a possible data bit transition, the whole search is performed on two consecutive data sets of 10 ms.

### 7.4.2 Tracking algorithms implementation

There is currently no data bit synchronization and decoding implemented, since all the ephemeris data and time information are obtained by means of the aiding receiver.

The code loop exploits a non-coherent early-minus-late envelope discriminator configured with a code replica spacing of three samples. Accordingly to Equation 2.40, it can be expressed as:

$$D_\tau = \frac{\overline{b} - 3}{\overline{b}} \cdot \frac{E - L}{E + L} \text{ [chips]} \tag{7.18}$$

The E and L values are obtained by non-coherently accumulating 20 consecutive correlation results (each coherently integrated over one millisecond). This guarantees the integrity of the DLL measurements, as they are not affected by a data bit transition. The code discriminator output is fed to the code filter, implemented as a standard second order lowpass, as illustrated in Figure 2.28. The filter is updated every $T = 0.02$ s and is configured with the parameters regrouped in Table 7.12.

| | | |
|---|---|---|
| $B_{nd}$ | $0.53 \cdot \omega_{0d}$ | 1Hz |
| $\omega_{0d}^2$ | - | 3.56 $\text{Hz}^2$ |
| $a_2\omega_{0d}$ | $1.414 \cdot \omega_{0d}$ | 2.67 Hz |
| $T_d$ | - | 0.02 s |

**Table 7.12:** DLL filter coefficients [Kap06]. $B_{nd}$ represents the noise bandwidth of the DLL loop.

The carrier loop is implemented as a second order PLL assisted by a first order FLL, as illustrated in Figure 2.27. The idea is to operate both the FLL and PLL discriminators continuously and integrate them in a single tracking loop. If the FLL error is zeroed, the filter becomes a pure second order PLL and vice versa. The implemented frequency and phase discriminators are described in Equation 2.35 and Equation 2.37 respectively. In order to accommodate the quartz short term variations, a small measurement time interval $t_2 - t_1$ is required for the frequency discriminator.

Consequently, both discriminators are operated by coherently accumulating two consecutive correlation results over two milliseconds. Several phase and frequency measurements are performed and averaged over 20 ms, so minimizing the influence of a possible data bit transition. The filter is updated every $T = 0.02$ s and is configured with the parameters summarized in Table 7.13.

| | | |
|---|---|---|
| $B_{nf}$ | $0.25 \cdot \omega_{0f}$ | 2Hz |
| $B_{np}$ | $0.53 \cdot \omega_{0p}$ | 12Hz |
| $\omega_{0f}$ | - | 8 Hz |
| $\omega_{0p}^2$ | - | 512.64 $\text{Hz}^2$ |
| $a_2\omega_{0p}$ | $1.414 \cdot \omega_{0p}$ | 32 Hz |
| $T_{pf}$ | - | 0.02 s |

**Table 7.13:** PLL-assisted-FLL filter coefficients [Kap06]. $B_{nf}$ and $B_{np}$ represent the noise bandwidths of the FLL and PLL loops respectively.

## 7.5 Performance of the implementation

The purpose of the following subsection is to validate the proposed receiver implementation in terms of position accuracy and CPU computational load. Note that the objective here is not to achieve the best accuracy or provide the highest sensitivity (the receiver is not optimized in that sense), but to demonstrate that the proposed solution works properly in real-time.

Three different signal sources are used for the accuracy tests:

- a single channel *Spirent GSS6100* simulator [Spi10], for testing specific algorithms of the receiver (useful for debugging purposes);
- a 12 channels *Spirent GSS8000* simulator [Spi10], for testing the receiver under controlled and reproducible conditions;
- a roof antenna, for testing the receiver with real satellites signals.

### 7.5.1 Static position (simulated signal)

The receiver is tested in static position under good sky view conditions, at the nominal power level, with the simulation parameters summarized in Table 7.14.

| Position | N 0°0′, E 0°0′, H 500 m |
|---|---|
| Simulation start time | 8.7.2009 00h00 |
| Simulation length | 1 h |
| Number of test runs | 20 |

**Table 7.14:** Parameters of the simulation for the static scenario.

The same scenario is repeated for the 20 runs and the receiver parameters are reset before each new run. The results are presented in Figure 7.8 (single run illustrated) and Figure 7.9 (20 averaged runs illustrated). Each figure is subdivided into four plots providing the following information:

1. a deviation map with the position error X (longitude) versus Y (latitude);
2. a histogram of the position error for X (longitude) and Y (latitude);
3. an integrated position error for X (longitude) and Y (latitude);
4. a position error for X (longitude) and Y (latitude) versus time. The plot also includes the number of satellites used by the PVT solution for computing the position (shown in green).

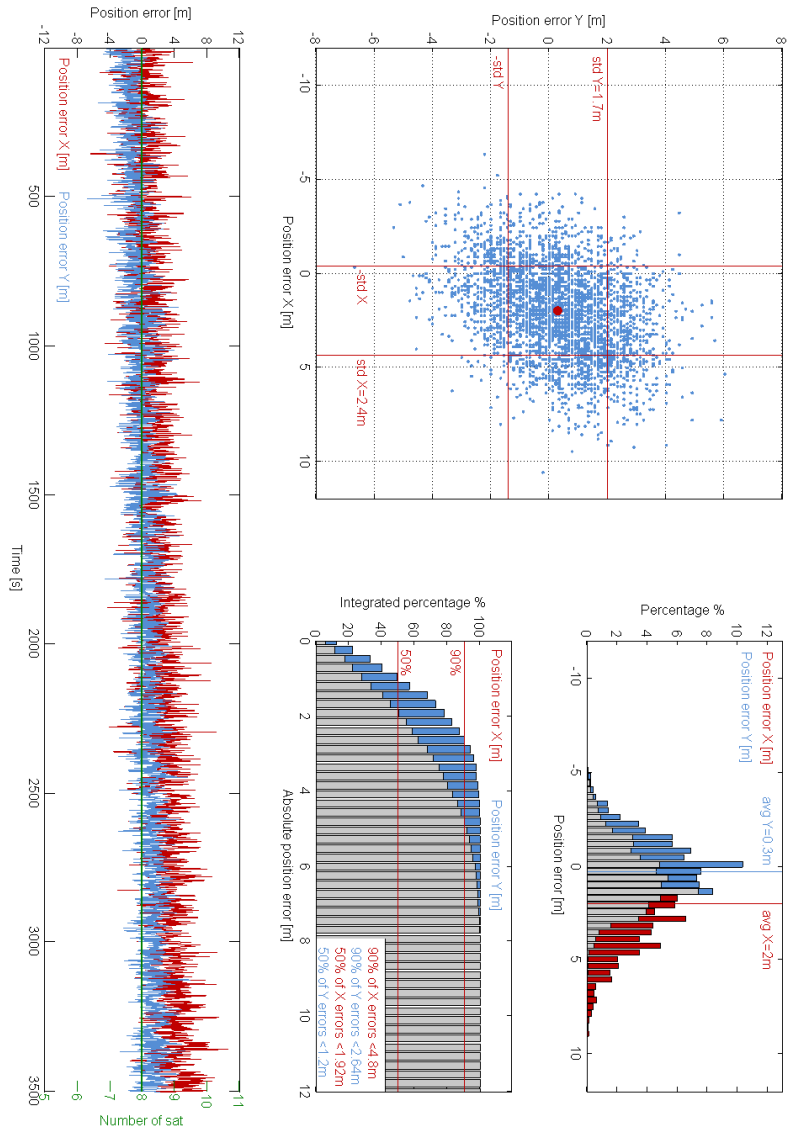


**Figure 7.8:** Static position with simulated signal (single run).

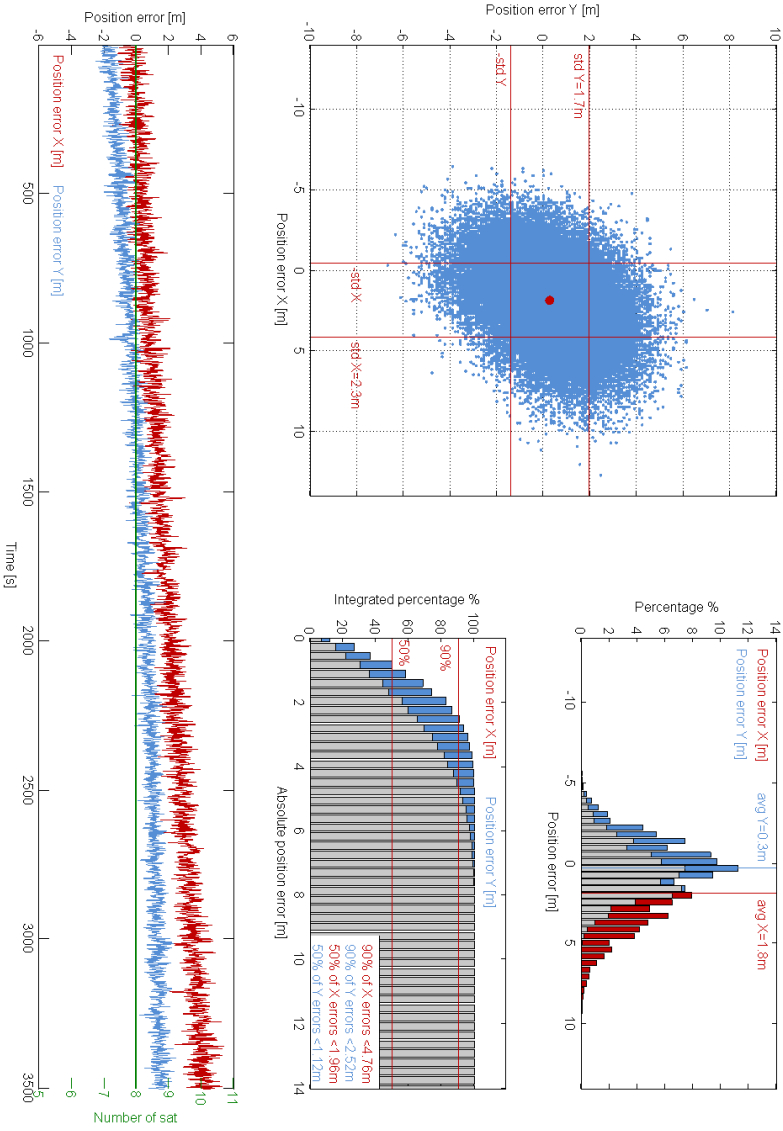


**Figure 7.9:** Static position with simulated signal (20 averaged runs).

Table 7.15 summarizes the different accuracy achievements:

| | Single run | | 20 averaged runs | |
|---|---|---|---|---|
| | X | Y | X | Y |
| Average error | 2 m | 0.3 m | 1.8 m | 0.3 m |
| Standard deviation error | 2.4 m | 1.7 m | 2.3 m | 1.7 m |
| Integrated error (50%) | <1.9 m | <1.2 m | <2 m | <1.1 m |
| Integrated error (90%) | <4.8 m | <2.6 m | <4.8 m | <2.5 m |

**Table 7.15:** Accuracy achievements for the static position (simulated signal).

The following observations can be concluded:

- the position error is kept within ±5 m in 90% of the time (over 20 hours of test). This corresponds to the accuracy that was expected from the PVT solution provided by *u-blox*;
- the results are reproducible over the different runs. For example, the distribution of the position error is very similar for a single run and for all the runs. This also applies for the evolution of the error versus time, where the trends are comparable in both figures;
- the positions are systematically affected by an offset of a few meters that is attributed to the initial geometry of the constellation. The satellites are not equally distributed in space and depending on their respective sky elevation, some may impact the position accuracy more than others;
- the position offset evolved over time, influenced by the changes of the constellation geometry. However, we assume here that the average offset over a 12-hour observation period tends to zero, as observed with the real signals tests in Subsection 7.5.2.

### 7.5.2 Static position (real signal)

The receiver is connected to a fixed antenna installed on the roof of the IMT building in Neuchâtel. The configuration of the test is summarized in Table 7.16.

| | |
|---|---|
| Position | N 46.99385°, E 6.9405°, H 450 m |
| First test start time | 1.11.2009 14h28 |
| Last test stop time | 2.11.2009 8h28 |
| Test length | 1 h |
| Number of test runs | 18 |

**Table 7.16:** Parameters of the tests for the static scenario.

Contrarily to the previous test with a simulated signal, the different runs are done sequentially during one hour, for a total duration of 18 hours. The receiver parameters are reset before each new run. The results are presented in Figure 7.10 (single run illustrated) and Figure 7.11 (18 averaged runs illustrated).

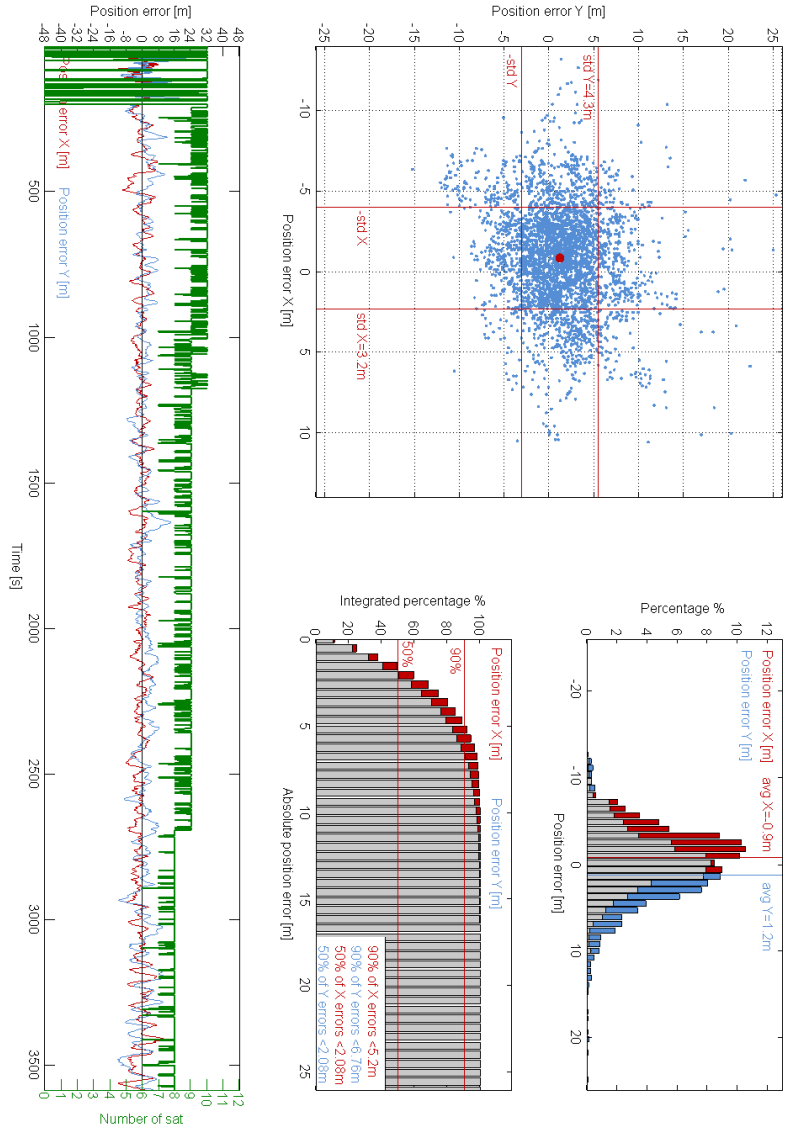


**Figure 7.10:** Static position with real signal (single run).

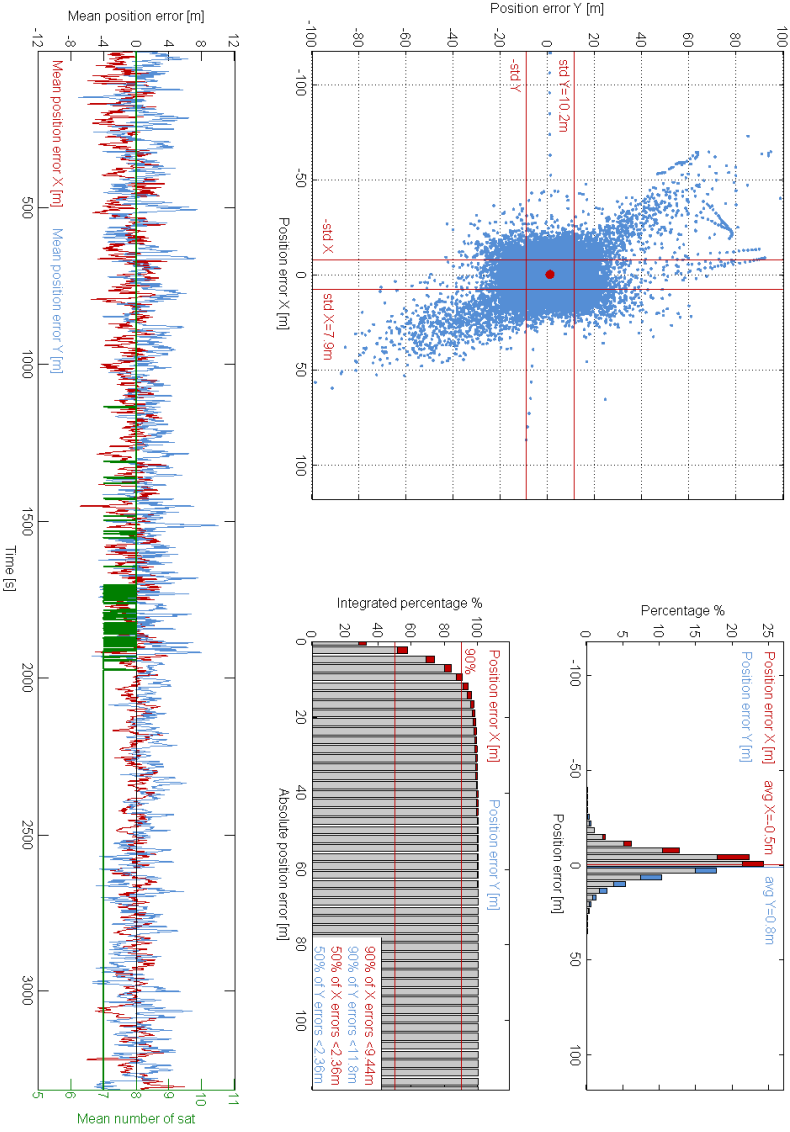


**Figure 7.11:** Static position with real signal (18 averaged runs).

Table 7.17 summarizes the different accuracy results:

| | Single run | | 18 averaged runs | |
|---|---|---|---|---|
| | X | Y | X | Y |
| Average error | 0.9 m | 1.2 m | 0.5 m | 0.8 m |
| Standard deviation error | 3.2 m | 4.3 m | 7.9 m | 10.2 m |
| Integrated error (50%) | <2.1 m | <2.1 m | <2.4 m | <2.4 m |
| Integrated error (90%) | <5.2 m | <6.8 m | <9.4 m | <11.8 m |

**Table 7.17:** Accuracy achievements for the static position (real signal).

The following observations can be concluded:

- the measurements are not reproducible, since they are affected by multi-path effect and temporary satellites signal masking;
- the multi-path effect, induced by the signal reflection on the surrounding buildings and on the lake, can affect some of the satellites signals (up to four satellites at the same time with measured code delays of a few tens of nanoseconds). Since no multi-path mitigation is performed within the receiver, the position accuracy may get strongly affected in the range of several tens of meters. This can be observed in Figure 7.11;
- when the multi-path effect is minimal such as in Figure 7.10, the position error is kept within ±7 m in 90% of the time;
- the position offset, influenced by the changes of the constellation geometry, gets averaged over the 18 observation periods and tends to zero, as illustrated in the error versus time plot of Figure 7.11.

### 7.5.3 Dynamic trajectory (simulated signal)

The receiver is tested in motion, following a square racetrack of 1.5 km side length with a maximal speed of 50 km/h (on cornering) and 100 km/h (on straight line). The simulation parameters are summarized in Table 7.18.

| | |
|---|---|
| Start position (bottom left corner) | N 46°59′, E 6°56′, H 500 m |
| Simulation start time | 24.9.2009 06h00 |
| Simulation length | 3600 s |
| Number of test runs | 10 |

**Table 7.18:** Parameters of the simulation for the dynamic trajectory.

The results are presented in Figure 7.12.

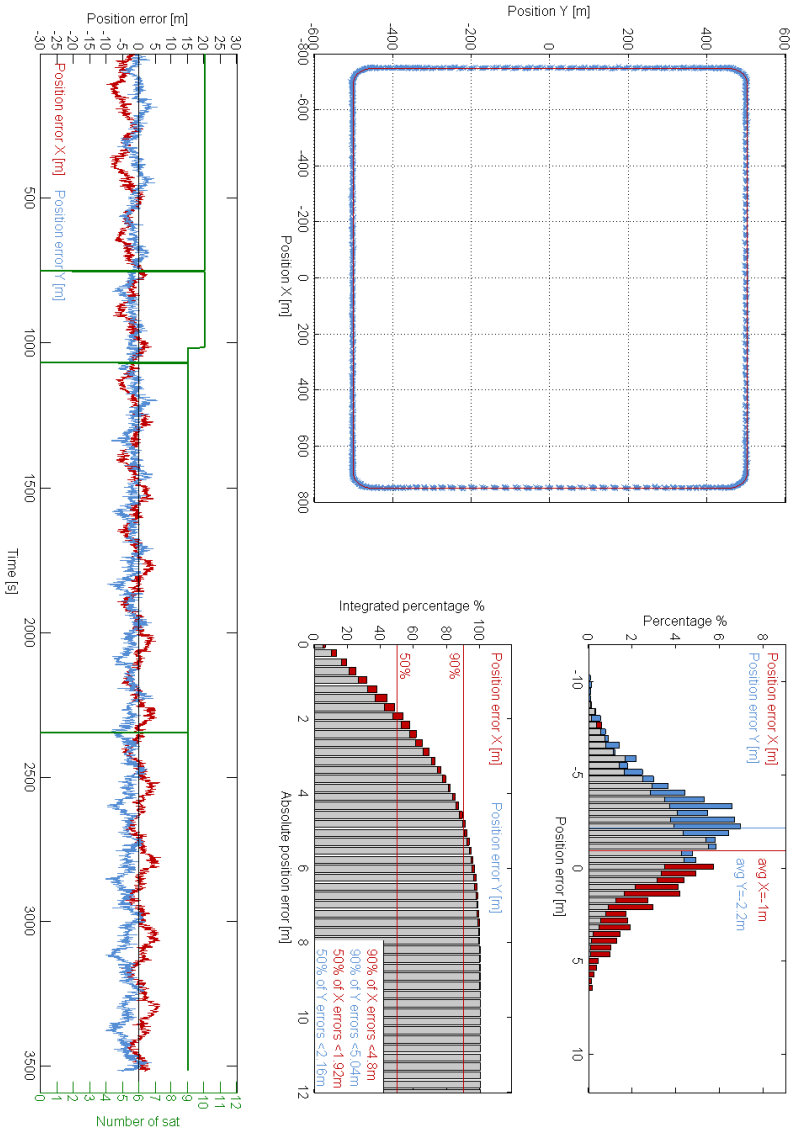


**Figure 7.12:** Dynamic trajectory with simulated signal (single run).

Table 7.19 summarizes the different results.

| | Single run | | 10 averaged runs | |
|---|---|---|---|---|
| | X | Y | X | Y |
| Average error | -1 m | -2.2 m | -1 m | -2.1 m |
| Standard deviation error | 2.8 m | 2.3 m | 2.7 m | 2.3 m |
| Integrated error (50%) | <1.9 m | <2.2 m | <2 m | <2.2 m |
| Integrated error (90%) | <4.8 m | <5 m | <4.8 m | <5 m |

**Table 7.19:** Second scenario.

The following observations can be concluded:

- the position error is kept within ±5 m in 90% of the time (over all the 10 test runs), which is comparable to the results obtained for the static tests;
- the largest errors are observed when the vehicle drives along the rounded corners of the rectangle. This translates into the "saw tooth" appearance of the time plot;
- the reference track was recorded on the *Spirent* simulator with a time resolution of 20 ms. The first 100 position fixes of the software receiver were best matched with the positions of the reference track to achieve time synchronization. For each subsequent position fix the absolute position error (in X and Y) was calculated with respect to the reference track.

### 7.5.4 High dynamic (simulated signal)

The next tests are performed with the *Spirent GSS6100* simulator on a single channel. The objective is to test the robustness of the tracking loops with respect to different high dynamic environments. The first test simply consists in switching the software receiver on after a long inactivity period (i.e. the prototype hardware components are at ambient temperature) and starting an instantaneous acquisition. The progressive heating of the board affects the quartz stability (which is not temperature compensated) and translates into a residual Doppler offset varying in the range of several kilohertz in less than one minute. Figure 7.13 shows respectively the filtered output of the PLL-assisted-FLL and DLL, and the in-phase and quadrature correlator outputs.

The second test is performed using the default high dynamic scenario *PROF1* of the simulator, consisting in a succession of acceleration and deceleration phases interrupted by constant velocity periods. The simulation parameters summarized in Table 7.20.

| | |
|---|---|
| Jerk amplitude | 20 m/$s^3$ (0.33 s) |
| Maximum acceleration | 6 m/$s^2$ |
| Period of constant acceleration | 1.1 s |
| Period of constant velocity | 1.1 s |

**Table 7.20:** Parameters of the *Spirent* default scenario *PROF1*.

The results are presented in Figure 7.13 and Figure 7.14.

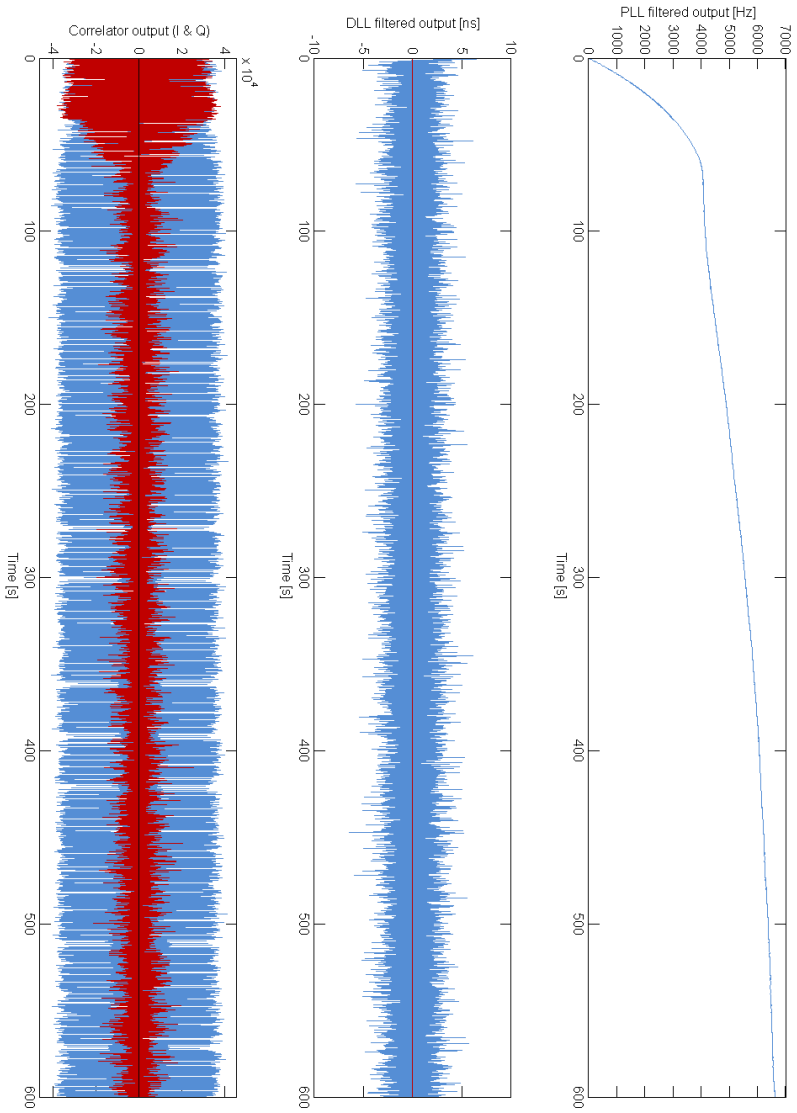


**Figure 7.13:** Doppler change due to the quartz heating. From top to bottom: 1) PLL filter output, 2) DLL filter output, 3) in-phase (blue) and quadrature (red) P correlator outputs.

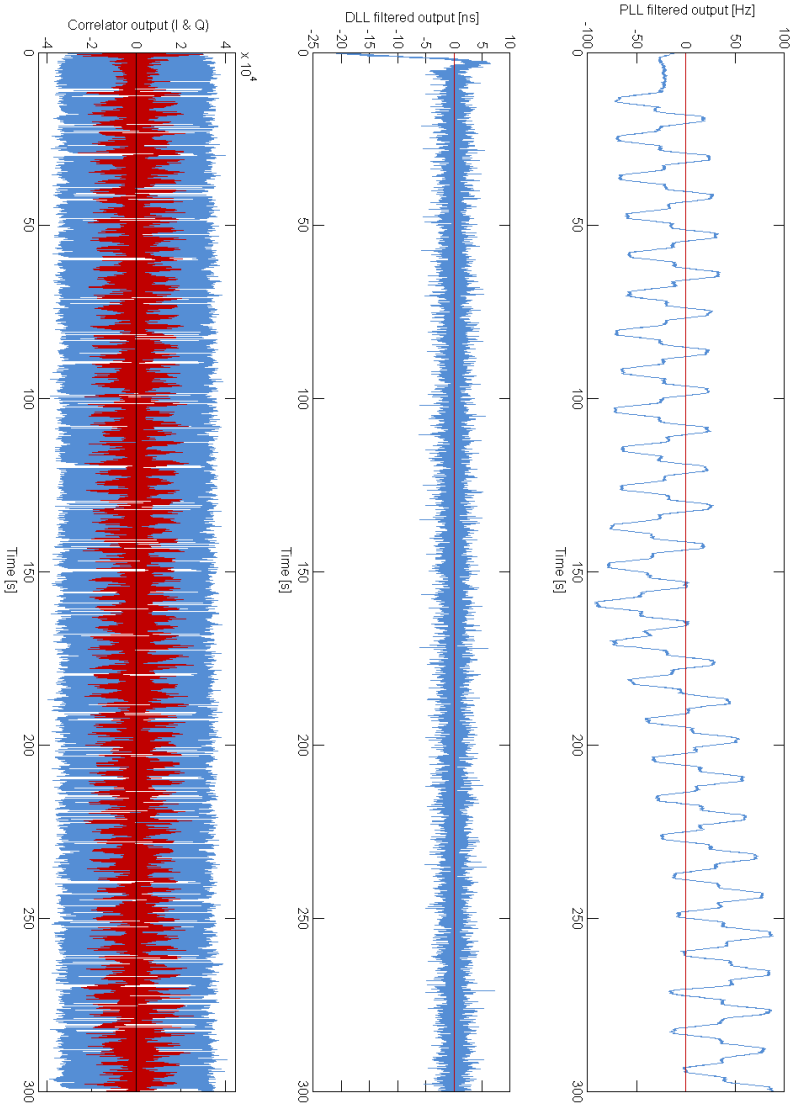


**Figure 7.14:** *Spirent* default scenario *PROF1* with a maximal acceleration of 6 m/$s^2$. From top to bottom: 1) PLL filter output, 2) DLL filter output, 3) in-phase (blue) and quadrature (red) P correlator outputs.

The software receiver can accommodate relatively high dynamic environments thanks to the short pre-detection time (2 ms) used in the carrier discriminators and thanks to the PLL-assisted-FLL implementation of the loop filter. For low dynamic environments, the signal lock is done in PLL mode only with a minimal assistance of the FLL. During acceleration period, the FLL assistance increases and becomes predominant in case of high dynamic environment. This can be observed in Figure 7.13, where a strong Doppler change ($\approx$ 100 Hz/s) occurs during the first 30 seconds, requiring a maximal assistance of the FLL to keep track of the satellite. The acceleration then strongly decreases ($\approx$ 10 Hz/s) and the signal gets progressively phase locked thanks to the PLL, which becomes predominant.

### 7.5.5 Post-processing time

The main goal of this test is to estimate the computational load of the software receiver by measuring the time needed to post-process one minute of data, stored on the PC hard drive, with a 100% CPU load. The whole data file (approximately 650 MB) is first transferred from the hard drive into the PC Random Access Memory (RAM) and the software receiver starts the processing flow and measures the time spent for both the acquisition and tracking. This procedure is repeated with 1 up to 12 satellite channels, on several platforms with different processor types and architectures. This test gives an estimation of the brute processing load and demonstrates the capability of the software to run in real-time mode on different CPU architectures. The host systems specifications are given in Table 7.21.
The acquisition takes place in two steps, by first searching the satellite over the full frequency uncertainty ($\pm$40 kHz) via the parallel frequency search. Once acquired, a re-acquisition is operated on specific frequency bins over the full code range via the parallel code search. Then the receiver switches into tracking mode. The results are presented in Figure 7.15, Figure 7.16, and Figure 7.17.

| PC model | *Dell Precision 380* | | *Dell Precision 380* | *Asus EeePC 1000H* |
|---|---|---|---|---|
| PC type | Desktop | Desktop | Desktop | Netbook |
| CPU model | *Intel Core 2 Duo E6700* | *AMD Athlon 64 3700+* | *Intel Pentium 4 640* | *Intel Atom N270* |
| CPU type | Dual core | Single core | Single core | Single core |
| CPU clock | 2667 MHz | 2210 MHz | 3200 MHz | 1600 MHz |
| CPU L1 cache | 32 kB per core | | 16 kB | 32 kB |
| CPU L2 cache | 4 MB | 2 MB | 2 MB | 512 kB |
| Hyperthreading | No | No | Yes | Yes |
| System memory | 2048 MB | 1024 MB | 1024 MB | 2048 MB |

**Table 7.21:** Specifications of the different host systems.

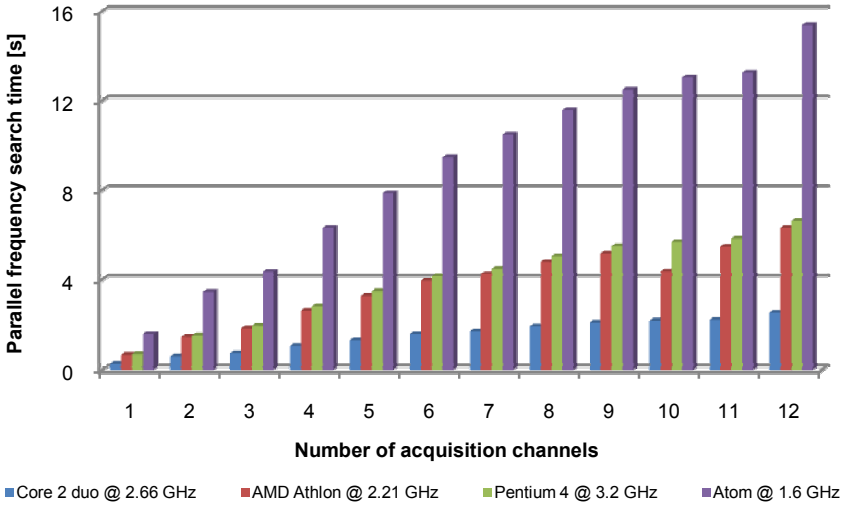


**Figure 7.15:** Time requested for the parallel frequency search in post-processing mode (±45 kHz Doppler uncertainty).

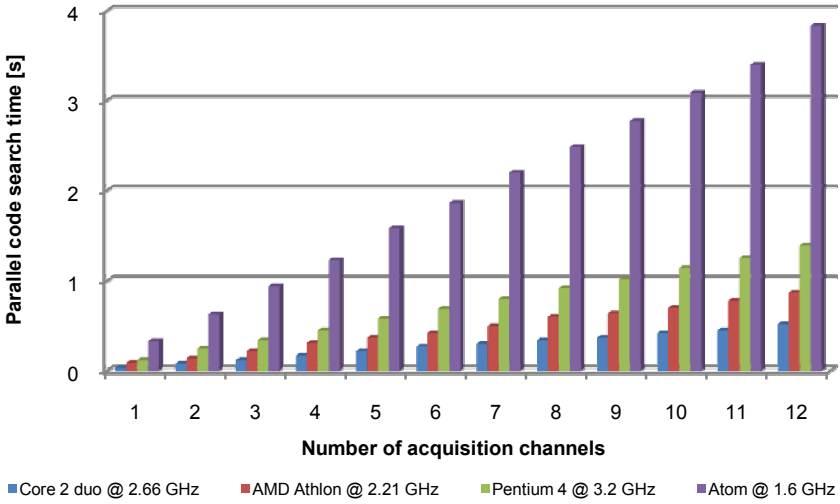


**Figure 7.16:** Time requested for the parallel code search in post-processing mode (seven Doppler bins).

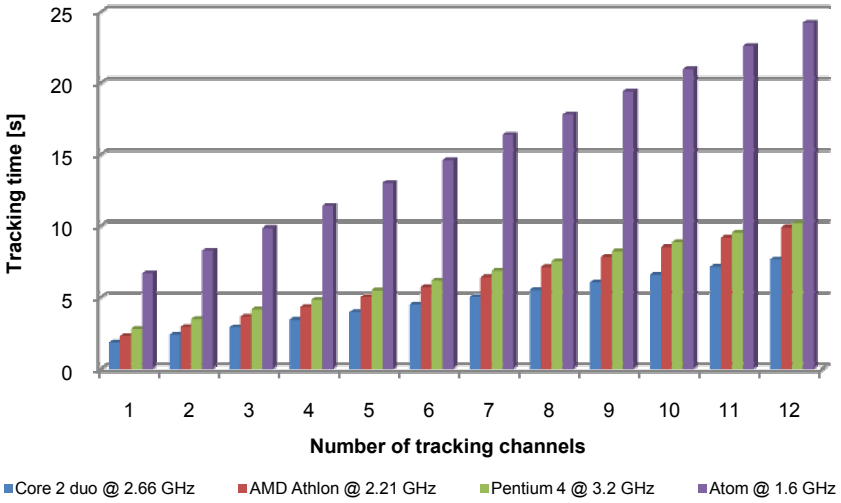


**Figure 7.17:** Time requested for tracking 60 seconds of data in post-processing mode.

Table 7.22 summarizes the different results of Figure 7.15, Figure 7.16, and Figure 7.17.

| | *Core 2 Duo* | *Athlon* | *Pentium 4* | *Atom* |
|---|---|---|---|---|
| Acquisition | 2.5 s | 6.3 s | 6.6 s | 15.4 s |
| Re-acquisition | 0.5 s | 0.9 s | 1.4 s | 3.8 s |
| Tracking | 7.6 s | 9.9 s | 10.2 s | 24.2 s |

**Table 7.22:** Post-processing time for the acquisition (parallel frequency search), re-acquisition (parallel code search) and tracking (60 s of data) of 12 satellites.

The following observations can be concluded:

- for the fastest configuration (*Core 2 Duo* with one single core used), the tracking of 12 satellites is operated in less than 8 seconds, making the execution more than seven times faster than real-time operations;
- for the slowest configuration (*Atom*), the tracking of 12 satellites is operated in less than 25 seconds. This confirms the extremely fast execution of the code and the efficiency of the implemented base-band architecture;
- the acquisition time of the parallel frequency search depends on the code phase of the incoming signal (since the algorithm sequentially searches for all the code phases until a satellite is found). On the other hand, the parallel code search is performed a fixed amount of times (on seven neighboring Doppler bins) and results in execution time proportional to the number of satellite channels;
- the code execution time strongly depends on the CPU architecture.

### 7.5.6 Processor load and memory requirements

The aim of this test is to measure the CPU load and the memory requirements of the receiver, when operating in real-time with the complete framework activated (USB, PVT solution, and display). The test is performed with ten concurrent satellite channels on three of the platforms of Table 7.21. The tool *ProcessExplorer* [Mic09] is used for measuring the CPU activity. The results are presented in Figure 7.18, Figure 7.19, and Figure 7.20.

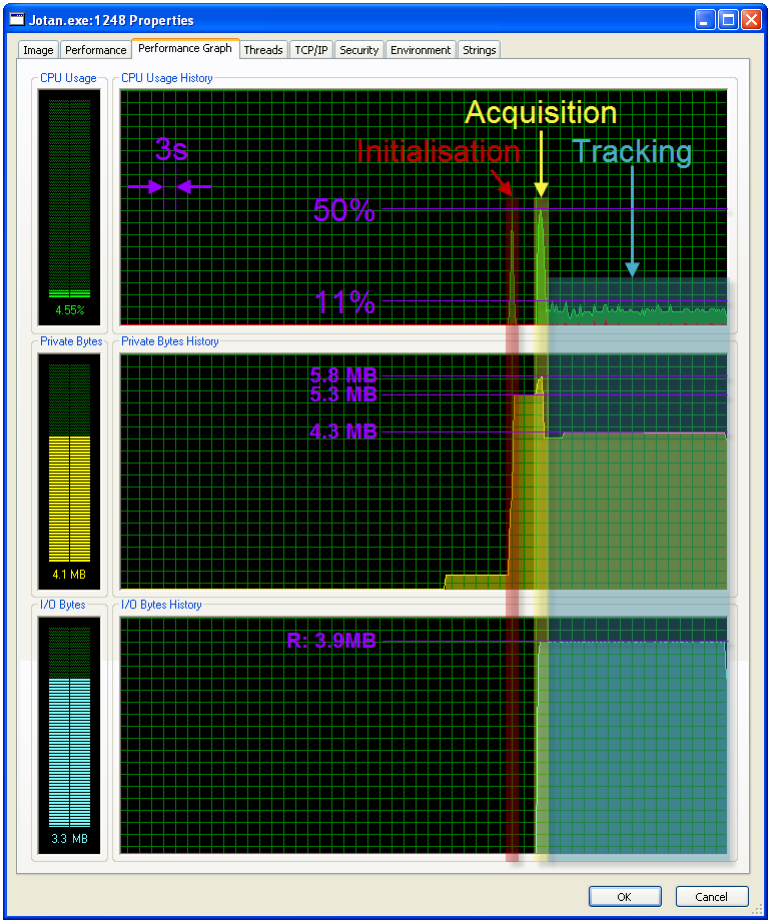


**Figure 7.18:** CPU load and memory requirements for the *Core 2 Duo* clocked at 2.66 GHz. Since the CPU is a dual core, a load of 50% represents one core fully sollicitated.

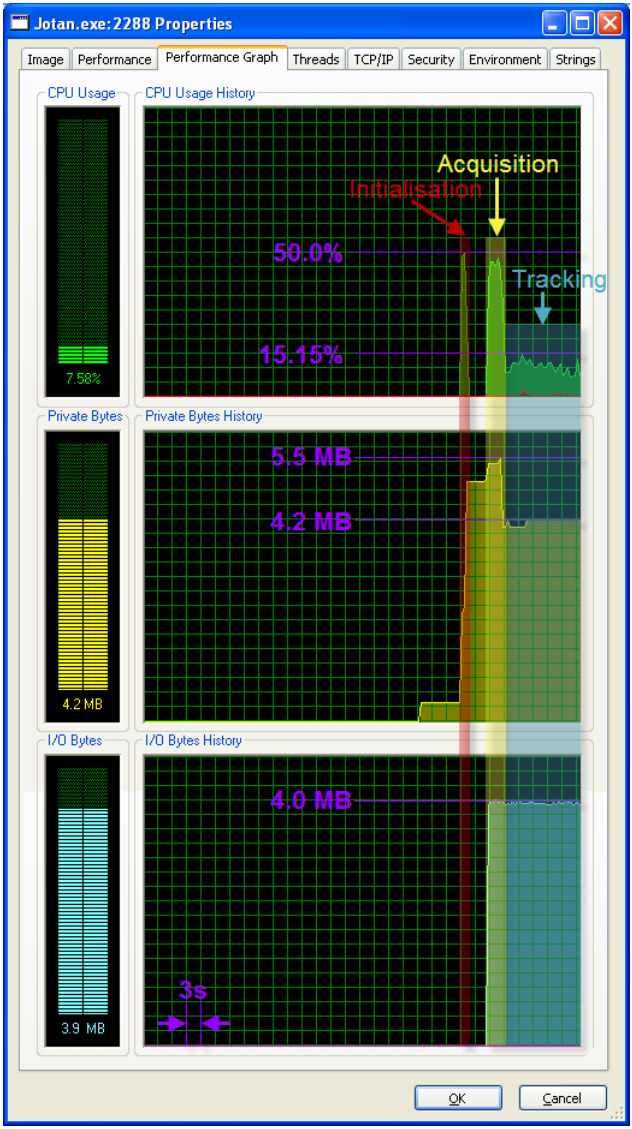


**Figure 7.19:** CPU load and memory requirements for the *Pentium 4* clocked at 3.2 GHz, with the hyper-threading functionality enabled.

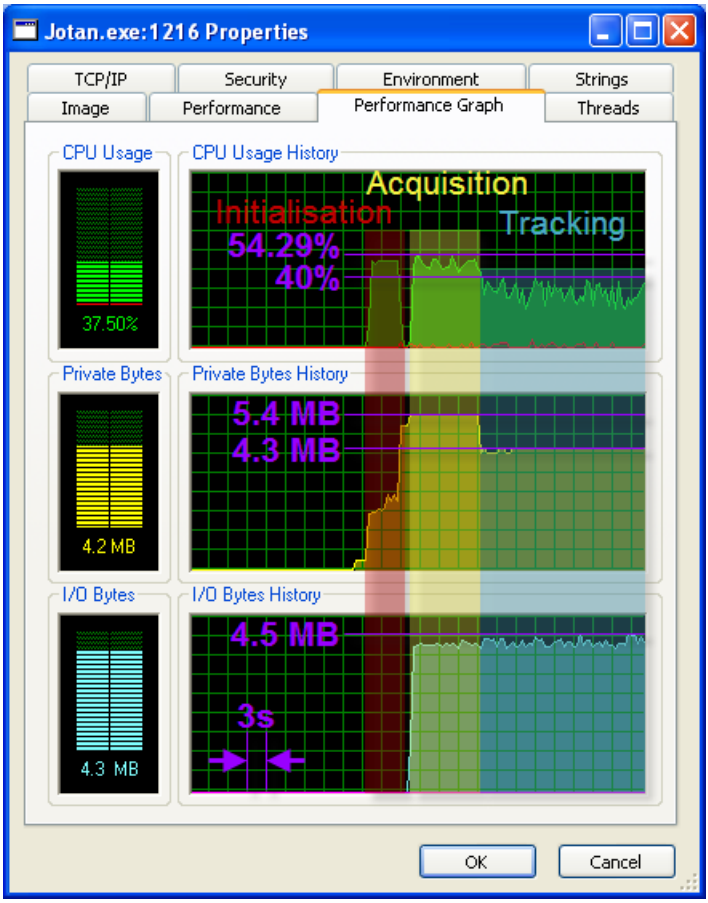


**Figure 7.20:** CPU load and memory requirements for the *Atom* clocked at 1.6 GHz, with the hyper-threading functionality enabled.

Table 7.23 summarizes the different results of Figure 7.18, Figure 7.19, and Figure 7.20.

| | *Core 2 Duo* | *Pentium 4* | *Atom* |
|---|---|---|---|
| Maximal CPU load | 22% | 15% | 40% |
| Averaged CPU load | 15% | 11% | 32% |
| Memory load | 4.3 MB | 4.2 MB | 4.3 MB |

**Table 7.23:** CPU and memory requirements for the concurrent tracking of ten satellites in real-time. The CPU activity of the *Core 2 Duo* is given for one single core. Both the *Pentium 4* and *Atom* have the hyper-threading functionality enabled.

The following observations can be concluded:

- for the fastest configuration (*Core 2 Duo*), on average 15% of one single core are solicited during the tracking of ten satellites;
- for the slowest configuration (*Atom*), on average 32% of the processor time are used during the tracking of ten satellites;
- the memory requirements are of a few megabytes, regardless of the configuration.

## 7.6 Summary

Chapter 7 describes the implementation of the proposed architecture in a demonstrator. The latter consists in a RF front-end board (for capturing, conditioning and digitizing the analog signal) connected via USB 2.0 to a standard *Windows* based PC running the software, as well as the PVT solution. The receiver is configured with a 3-bit complex input data stream sampled at 8 MHz. This setup was tested with different signal sources on several platforms equipped with various microprocessors. With simulated satellites signals (generated with a *Spirent GSS8000* simulator under clear sky view), the position accuracy is better than ±5 m in 90% of the time and the results are reproducible over the 20 hours test period.

On the other hand, most of the measurements performed with real signals (obtained from an external roof antenna) are affected by multipath. Since no mitigation is performed within the receiver, the position sometimes gets corrupted up to several tens of meters depending on the geometry of the satellites constellation. In the best cases, the accuracy is higher than $\pm 7$ m in 90% of the time (one hour observation period), while over the 18 consecutive hours of test, the position error is kept within $\pm 12$ m in 90% of the time. Regarding the computational load, the concurrent tracking of ten satellites occupies less than 22% of the CPU time of one single core of a *Intel Core 2 Duo E6700* microprocessor (clocked at 2.66 GHz). The memory requirements are limited to a few megabytes. This demonstrates the feasibility of implementing a real-time software receiver on various independent platforms and validates the proposed architecture.

# Chapter 8

# Conclusion

The aim of a software receiver is to substitute the dedicated hardware chip by a programmable microprocessor, in order to process the digital data stream in software. With modern mobile devices, such as PDAs and smartphones embedding powerful microprocessors, the complete receiver can be integrated with very few external components. The necessary hardware part is reduced to the minimum (i.e. a RF front-end responsible for the analog signal conditioning and digitization) and most of the system resources, such as the processor and the memory, can be shared with the host. The interest clearly lies in the low cost opportunity, but also in the flexibility, since any low-level receiver functionality modification can be operated by a simple firmware update. Such capabilities become even more important as the world of global navigation is in complete effervescence, with the introduction of new satellites constellations (GLONASS, Galileo, Compass, ...) foreseen within the next few years. The users of software receivers will derive full benefit from this next generation of signals, with a simple software upgrade and without purchasing new hardware components.

However, the implementation of the receiver in software introduces new constraints as compared to hardware based solutions. The major issue is the large computing resources required for the digital signal processing. A straightforward transposition of the traditional architectures into software leads to an amount of integer operations which is not suitable for today's fastest computers. Consequently, new approaches have to be considered, in order to significantly reduce the computational load of the operations involved in the receiver processing chain.

Many solutions proposed in the literature rely on the use of advanced CPU instructions for improving the efficiency and speed of the processing, but are tied to specific configurations that severely limit their portability. On the other hand, the use of bitwise processing exploiting universal logical operations still suffers from a lack of flexibility. It requires the front-end output stream to be specifically formatted, while any change in the signals structure implies a significant adaptation of the code and impacts the complexity. Consequently, in order to implement a software receiver operating in real-time on different platforms and accommodating various signals configurations, there is a need for developing a new architecture combining both flexibility and efficiency.

## 8.1 Achievements of the thesis

This thesis work introduces a completely new receiver architecture based on batch processing of the incoming samples. The basic idea consists in restricting the frequencies range of the carriers internally generated to a few kilohertz, so that their time evolution becomes very slow as compared to the sampling frequency. Since the carrier magnitude remains constant during hundreds or thousands of clock cycles, data appertaining to the same carrier interval are accumulated into batches that are multiplied only once by the carrier value, instead of multiplying each data separately. The same also applies for the code removal, where samples appertaining to the same chip are summed up and multiplied only once with the chip value. The batch processing also simplifies the code and carrier synthesis, as they now consist in predicting the times where the carrier, respectively the chips, transitions occur. Both carrier and code generator can be operated at a low frequency, proportional to the residual Doppler and the code frequency respectively, thus allowing the real-time generation of the signals waveforms continuously and at any desired frequency. This way, the Doppler offset can be compensated natively during the carrier and code generation process, which greatly simplifies the integration of the blocks in the receiver, as they will easily accommodate conventional tracking schemes. Finally, the memory requirements are reduced to the strict minimum as no pre-generated sequences need to be stored.

The key point of the proposed solution relies on the progressive data throughput reduction, which is first decreased from the sampling rate of several megahertz to the code rate of one megahertz, and finally to a frequency proportional to the number of carrier phases. As compared to a traditional hardware based receiver, where all the operations are performed at the sampling frequency, the amount of integer additions and multiplications involved in the base-band processing is reduced by almost one order of magnitude. Since most of the operations, such as the carrier and code synthesis, the carrier and code mixing, and the accumulation are preformed at a relatively low frequency with respect to the nominal sampling rate, the architecture can easily accommodate incoming signals sampled at high frequencies. Furthermore, as the data are systematically manipulated as integers (i.e. no bitwise processing is performed on the data), the signal and carrier quantization can change without significant code modification. This makes the proposed architecture extremely flexible and well suited for research and development.

The architecture was implemented in a demonstrator consisting in a RF front-end board (for capturing, conditioning and digitizing the signal) connected via USB 2.0 to a standard *Windows* based PC running the receiver software, as well as the PVT solution. The receiver is configured with a 3-bit complex input data stream sampled at 8 MHz. This setup was successfully tested with various signal sources (simulated and real ones), on four host computers equipped with different microprocessors. With simulated satellites signals (generated with a *Spirent GSS8000* simulator under clear sky view), the position accuracy is better than $\pm 5$ m in 90% of the time and the results are reproducible over the 20 hours test period. On the other hand, most of the measurements performed with real signals (obtained from an external roof antenna) are affected by multipath. Since no mitigation is performed within the receiver, the position sometimes gets strongly corrupted depending on the geometry of the satellites constellation. However, when the effect is minimal, the accuracy is higher than $\pm 7$ m in 90% of the time for one hour observation period.

The brute computational load of the receiver was estimated by post-processing 60 seconds of recorded data under a 100% CPU load. The concurrent tracking of 12 satellites on an *Intel Core 2 Duo E6700* (clocked at 2.66 GHz) is performed in less than 8 seconds with one single core used, making the execution seven times faster than the real-time operations.

The same procedure required 24 seconds on an *Intel Atom N270* (clocked at 1.6 GHz). The CPU load and the memory requirements of the system were analyzed by operating the receiver in real-time with the complete framework (USB, PVT solution and display) activated. The concurrent tracking of 10 satellites occupies on average 15% of the CPU time of one single core of an *Intel Core 2 Duo E6700* microprocessor. The memory requirements are limited to a few megabytes only. This demonstrates the capability of the receiver to operate in real-time on different platforms with a modest impact on the CPU load.

## 8.2 Future steps

- the ephemeris data and time information are provided by the aiding receiver. There is currently no data synchronization and decoding implemented, making the standalone functioning of the software receiver not possible. This also affects the tracking sensitivity, since the absence of data bit synchronization restricts the coherent integration time to one or a few milliseconds. Implementing an algorithm for the data bit detection would thus allow the use of longer integration times and consequently improve the overall receiver sensitivity;

- the receiver operates either in acquisition or in tracking mode. There is currently no way for a single satellite channel to switch back from tracking to acquisition, making the re-acquisition of a satellite or the acquisition of a new one not possible;

- the receiver currently processes the GPS L1 CA signal only. However, the base-band architecture can be modified in order to accommodate the next generation of satellites signals, such as Galileo E1 or Galileo E5;

- in the perspective of developing a platform-independent receiver, the software was voluntary not optimized to take advantage of functionalities provided by modern processors (such as hyper-threading, SIMD or multi cores capabilities). Tailoring the code for a specific application would definitively result in a significant performance gain, making the proposed architecture even more efficient.

# Bibliography


[Act10] Actel corporation. *Designing FIR filters with Actel FPGAs*. Application note AC120, 2010. `http://www.actel.com/documents/FIR_Filters_AN.pdf`.

[Ako96] D. Akos and J. B. Tsui. "Design and implementation of a direct digitization GPS receiver front end". *IEEE Transactions on Microwave Theory and Techniques*, 44(12):2334–2339, 1996.

[Ako97] D. Akos. *A software radio approach to Global Navigation Satellite System receiver design*. Ph.D. thesis, University of Ohio, 1997.

[Ako99] D. Akos, M. Stockmaster, J. B. Tsui, and J. Caschera. "Direct bandpass sampling of multiple distinct RF signals". *IEEE Transactions on Communications*, 47(7):983–988, 1999.

[Ako01a] D. Akos, P.-L. Normark, P. Enge, A. Hansson, and A. Rosenlind. "Real-time GPS software radio receiver". *2001 National Technical Meeting of the Institute of Navigation*, pp. 809–816. Long Beach, CA, USA, January 22-24 2001.

[Ako01b] D. Akos, P.-L. Normark, A. Hansson, A. Rosenlind, C. Stahlberg, and F. Svensson. "Global positioning system software receiver (gpSrx) Implementation in low cost/power programmable processors". *14th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 2851–2858. Salt Lake City, UT, USA, September 11-14 2001.

[Alt08] Altera corporation. *Altera Cyclone II EP2C35*, 2008. `http://www.altera.com/products/devices/cyclone2/cy2-index.jsp`.

[And99] R. Andraka and A. Berkun. "FPGAs make a radar signal processor on a chip a reality". *The 33rd Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 559–563. Pacific Grove, CA, USA, October 24-27 1999.

[Bar09] M. Baracchi, G. Waelchli, C. Botteron, and P.-A. Farine. "Realtime GNSS software receiver: challenges, status and perspectives". *GPS World*, 20(9):40–47, 2009.

[Bar10] M. Baracchi, G. Waelchli, C. Botteron, and P.-A. Farine. "Realtime GNSS software receiver: challenges, status and perspectives". *My Coordinates*, VI(5):7–9, 2010.

[Bee72] M. Beeler, R. Gosper, and R. Schroeppel. "Count ones - Item 169". *HAKMEM - MIT AI Memo*, 239, 1972.

[Bor07] K. Borre, D. M. Akos, N. Bertelsen, and S. H. Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach.* Applied and Numerical Harmonic Analysis. Birkhaeuser, Boston, MA. ISBN 0-8176-4390-7, 2007.

[Bos04] V. Bose. "A software driven approach to SDR design". *COTS: The Journal of Military Electronics and Computing*, January 2004.

[Bro07] A. Brown, B. Mathews, and D. Nguyen. "GPS/INS/STAR tracker navigation using a software defined radio". *29th Annual AAS Guidance and Control Conference*, vol. 128, pp. 111–122. Breckenridge, CO, USA, February 4-8 2007.

[Bue09] C. Buergi, G. Waelchli, and M. Baracchi. "A method of processing a digital signal derived from a direct-sequence spread spectrum signal and a receiver for carrying out the method". European Patent 09405207.3, November 2009.

[Bue10] C. Buergi, G. Waelchli, and M. Baracchi. "A method of processing a digital signal derived from a direct-sequence spread spectrum signal and a receiver". US Patent 12/694,145, January 2010.

[Cha06] S. Charkhandeh, M. Potovello, R. Watson, and G. lachapelle. "Implementation and testing of a real-time software-based GPS receiver for x86 processors". *2006 National Technical Meeting of the Institute of Navigation*, vol. 2, pp. 927–934. Monterey, CA, USA, January 18-20 2006.

[Cha07] S. Charkhandeh. *X86-based real-time L1 GPS software receiver*. Ph.D. thesis, University of Calgary, 2007.

[Cyp06] Cypress Semiconductor corporation. *EZ-USB FX2LP USB Microcontroller Datasheet*. Document number: 38-08032, Rev. K, 2006. `http://www.cypress.com`.

[Die95] A. J. v. Dierendonck. *Global Positioning System: Theory and Applications Volume I*, vol. 163 of *Progress in Aeronautics and Astronautics*. American Institute of Aeronautics and Astronautics, Inc., 370 L'Enfant Promenade, SW, Washington, DC. ISBN 1-56347-106-X, 1995.

[Eur10a] European Commission. *European GNSS (Galileo) Open Service Signal In Space Interface Control Document*, 2010. `http://ec.europa.eu/enterprise/policies/space/galileo/open-service/index_en.htm`.

[Eur10b] European Space Agency. *Galileo services*, 2010. `http://www.esa.int`.

[FPG10] FPGA-DEV. *Altera Cyclone II EP2C35 Entwicklungsboard v1.1*, 2010. `http://www.fpga-dev.de`.

[Fri07] M. Frigo and S. G. Johnson. "FFTW: Fastest Fourier Transform in the West", 2007. `http://www.fftw.org`.

[Gan04] S. Ganguly. "Real-time dual frequency software receiver". *Position Location and Navigation Symposium*, pp. 366–374. Monterey, CA, USA, April 26-29 2004.

[Gol67] R. Gold. "Optimal binary sequences for spread spectrum multiplexing". *IEEE Transactions on Information Theory*, 13(4):619–621, 1967.

[Hec04] G. W. Heckler and J. L. Garrison. "Architecture of a reconfigurable software receiver". *17th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 947–955. Long Beach, CA, USA, September 21-24 2004.

[Hec06] G. W. Heckler and J. L. Garrison. "SIMD correlator library for GNSS software receivers". *GPS Solutions*, 10(4):269–276, 2006.

[Hum06] T. E. Humphreys, M. L. Psiaki, and P. M. K. Jr. "GNSS receiver implementation on a DSP: status, challenges, and prospects". *19th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 2370–2382. Fort Worth, TX, USA, September 26-29 2006.

[ike10] ikeGPS by Surveylab. *GPS TTFF and startup modes*, 2010. `http://www.ikegps.com/downloads/TTFFstartup.pdf`.

[Int97] Intel corporation. *Pentium overdrive processor with MMX technology for Pentium processor-based systems*, 1997. `http://www.intel.com/design/archives/Processors/mmx`.

[Int09] Intel corporation. *Intel 64 and IA-32 architectures optimization reference manual*, 2009. `http://www.intel.com/products/processor/manuals`.

[iSu08] iSuppli Market Research. *Cell phones to overtake PNDs by 2011*, 2008. `http://www.isuppli.com`.

[Kap06] E. D. Kaplan and C. Hegarty. *Understanding GPS: Principles and Applications*. Artech House Mobile Communications Series. Artech House, Inc., Boston, 2nd ed. ISBN 1-58053-894-0, 2006.

[Lac95] R. J. Lackey and D. W. Upmal. "Speakeasy - the Military Software Radio". *IEEE Communications Magazine*, 33(5):56–61, 1995.

[Led03] B. M. Ledvina, S. P. Powell, P. M. Kintner, and M. L. Psiaki. "A 12-channel real-time GPS L1 software receiver". *2003 National Technical Meeting of the Institute of Navigation*, pp. 767–782. Anaheim, CA, USA, January 22-24 2003.

[Led04] B. M. Ledvina, M. L. Psiaki, S. P. Powell, and P. M. Kintner. "Bit-wise parallel algorithms for efficient software correlation applied to a GPS software receiver". *IEEE Transactions on Wireless Communications*, 3(5):1469–1473, 2004.

[Led06a] B. M. Ledvina, M. L. Psiaki, T. E. Humphreys, S. P. Powell, and P. M. Kintner. "A real-time software receiver for the GPS and Galileo L1 signals". *19th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 2321–2333. Fort Worth, TX, USA, September 26-29 2006.

[Led06b] B. M. Ledvina, M. L. Psiaki, S. P. Powell, and P. M. Kintner. "Real-time software receiver". US Patent US2006/0227856 A1, October 2006.

[Lin06] W. H. Lin, W. L. Mao, H. W. Tsao, F. R. Chang, and W. H. Huang. "Acquisition of GPS software receiver using split-radix FFT". *2006 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 4608–4613. Taipei, Taiwan, October 8-11 2006.

[Mar03] N. Martin, V. Leblond, G. Guillotel, and V. Heiries. "BOC(x,y) signal acquisition techniques and performances". *16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 188–198. Portland, OR, USA, September 9-12 2003.

[Mat88] P. G. Mattos. "A low-cost hand-held GPS navigation system receiver". *Fourth International Conference on Satellite Systems for Mobile Communications and Navigation*, pp. 217–221. London, UK, October 17-19 1988.

[Mat03] H. Mathis, P. Flammant, and A. Thiel. "An analytic way to optimize the detector of a post-correlation FFT acquisition algorithm". *16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 689–699. Portland, OR, USA, September 9-12 2003.

[Mic09] Microsoft Corporation, Mark Russinovich. *Process Explorer*, 2009. `http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx`.

[Mit92] J. Mitola. "Software radios survey, critical evaluation and future directions". *National Telesystems Conference 92*, pp. 15–23. Washington, DC, USA, May 19-20 1992.

[Mon07] C. Mongrédien, M. E. Cannon, and G. Lachapelle. "Performance evaluation of a GPS L5 software receiver using a hardware simulator". *European Navigation Conference on GNSS 07*. Geneva, Switzerland, May 29 - June 1 2007.

[Nor04] P.-L. Normark and C. Stahlberg. "Spread spectrum signal processing". International Patent WO2004/036238 A1, April 2004.

[Pan03] T. Pany, S. W. Moon, M. Irsigler, B. Eissfeller, and K. Fürlinger. "Performance assessment of an under sampling SWC receiver for simulated high-bandwidth GPS/Galileo signals and real signals". *16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 103–116. Portland, OR, USA, September 9-12 2003.

[Pen10] Pentek incorporated. *Putting undersampling to work*. Tutorials, 2010. `http://www.pentek.com/pildocs/6982/techother/putundersamp.pdf`.

[Pet06] M. G. Petovello and G. Lachapelle. "An efficient new method of Doppler removal and correlation with application to software-based GNSS receivers". *19th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 2407–2417. Fort Worth, TX, USA, September 26-29 2006.

[Pet08] M. G. Petovello and G. Lachapelle. "Methods and systems for doppler frequency shift removal and correlation for software-based receivers". US Patent US2008/0007448 A1, 2008.

[Pub10] Public Safety and Homeland Security Bureau. *Enhanced 9-1-1 - Wireless Services*, 2010. `http://www.fcc.gov/pshs/services/911-services/enhanced911/Welcome.html`.

[Pur05] P. Puricer, P. Kovar, L. Seidl, and F. Vejrazka. "GNSS software receiver - a versatile platform for navigation systems signals processing". *47th International Symposium ELMAR*, pp. 249–252. Zadar, Croatia, June 8-10 2005.

[Rak10] Rakon corporation. *Datasheet*, 2010. `http://www.rakon.com/Products/Pages/TCXOs.aspx`.

[Raz98] B. Razavi. *RF microelectronics*. Prentice Hall Communications Engineering and Emerging Technologies Series. Prentice Hall, Upper Saddle River, NJ, USA. ISBN 0-13-887571-5, 1998.

[Sar80] D. V. Sarwate and M. B. Pursley. "Crosscorrelation properties of pseudorandom and related sequences". *Proceedings of the IEEE*, 68(5):593–619, 1980.

[Sch96] B. Schaller. "The origin, nature, and implications of Moore's law: the benchmark of progress in semiconductor electronics". Microsoft corporation, September 26 1996. `http://research.microsoft.com/en-us/um/people/gray/Moore_Law.html`.

[Sch02] J. J. Schamus, J. B. Tsui, and D. M. Lin. "Real-time software GPS receiver". *15th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 2561–2565. Portland, OR, USA, September 24-27 2002.

[Spi10] Spirent Communications corporation. *Product overview*, 2010. `http://www.spirent.com/Positioning-and-Navigation/What_is_GPS_Simulation.aspx`.

[Sti05] J. Stillwell. *The Four Pillars of Geometry*. Springer Science+Business Media. Springer, 233 Spring Street, New-York, NY, USA. ISBN 0-387-25530-3, 2005.

[Tia08] J. Tian, W. Ye, S. Lin, and Z. Hua. "SDR GNSS receiver design over stand-alone generic TI DSP platform". *10th International Symposium on Spread Spectrum Techniques and Applications*, pp. 37–41. Bologna, Italy, August 25-28 2008.

[Tse02] C. H. Tseng. "Bandpass sampling criteria for nonlinear systems". *IEEE Transactions on Signal Processing*, 50(3):568–577, 2002.

[Tsu05] J. B. Tsui. *Fundamentals of Global Positioning System Receivers: A Software Approach*. Wiley Series in Microwave and Optical Engineering. John Wiley & Sons, Inc., New Jersey, 2nd ed. ISBN 0-471-70647-7, 2005.

[U-b10] U-blox corporation. *Product overview*, 2010. `http://www.u-blox.com/en/gps-modules.html`.

[US 95] US Department of Homeland Security. *Global Positioning System standard positioning service signal specification*, 1995. `http://pnt.gov/public/docs/1995/signalspec1995.pdf`.

[Wae07] G. Waelchli, G. Zamuner, D. Manetti, M. Frei, F. Chastellain, E. Firouzi, C. Botteron, P.-A. Farine, and P. Brault. "Development, implementation and validation of a real-time Galileo E1 signal acquisition and tracking scheme". *European Navigation Conference on GNSS 07*, pp. 626–634. Geneva, Switzerland, May 29 - June 1 2007.

[Wae09a] G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Performances of a new correlation algorithm for a platform-independent GPS software receiver". *2009 International Technical Meeting of the Institute of Navigation*, pp. 1062–1067. Anaheim, CA, USA, January 26-28 2009.

[Wae09b] G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Real-time carrier generation for a GNSS software receiver". *International Symposium on GPS/GNSS 2009*. Jeju, South Korea, November 4-9 2009.

[Wae10] G. Waelchli, M. Baracchi, C. Botteron, and P.-A. Farine. "Distributed arithmetic for efficient base-band processing in real-time GNSS software receivers". *Hindawi, Journal of Electrical and Computer Engineering*, January, 2010.

[War98] P. W. Ward. "Performance comparisons between FLL, PLL and a novel FLL-assisted-PLL carrier tracking loop under RF interference conditions". *11th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 783–795. Nashville, TN, USA, September 15-18 1998.

[Yan03] C. Yang. "Tracking of GPS code phase and carrier frequency in the frequency domain". *16th International Technical Meeting of the Satellite Division of the Institute of Navigation*, pp. 628–637. Portland, OR, USA, September 9-12 2003.

[Zog09] J.-M. Zogg. *GPS Essentials of Satellite Navigation*. U-blox AG. ISBN 978-3033021396, 2009.

# Acknowledgments

This PhD thesis could not have been successfully completed without the help and support of many people, who are all kindly acknowledged.

First, I wish to express my sincere gratitude to Prof. Fausto Pellandini, for offering me the opportunity to join his research team at IMT. I am also very grateful to his successor, Prof. Pierre-André Farine, for being my thesis supervisor and letting me work in the most favorable conditions.

My warm thanks go to my project leader and thesis co-supervisor, Cyril Botteron, for its availability during this work, and for his very constructive remarks. In particular, I thank him for the time he spent for reviewing my papers and this document.

My thanks go to all the team of *u-blox* in Thalwil for the enriching collaboration we had during this project. In particular, I would like to warmly thank Clemens Buergi for his continuous support.

Directly connected to this thesis, I am very much obliged to Marcel Baracchi, whom I collaborated with in the course of several projects in the field of GNSS. It was a real pleasure to work with you.

Working at IMT is a real privilege. I would like to thank all my friends, colleagues and ex-colleagues from ESPLAB, for the very fruitful collaborations in the framework of several projects, and all the great times we had during the ski week-ends and the countless parties.

I am very thankful to my family and friends for being very supportive and understanding during the finalization of this thesis. I am deeply grateful to my parents and my brother, for their trust and indefectible support.

*Finally you, my one and only, I just want to say I love you.*

# Curriculum Vitae


Name: Grégoire Waelchli
Birthdate: September 15, 1977
Address: Stand 26, 2502 Bienne, Switzerland
Nationality: Swiss citizenship


## Career description

**Ecole polytechnique fédérale de Lausanne - Neuchâtel since 2009**
*Research and development, PhD student*

- I developed a real-time GPS software receiver (industrial project with u-blox AG, Thalwil). This project lead to an international patent.

**Institute of microtechnolgy (IMT) - Neuchâtel 2002-2008**
*Research and development*

- I developed a Galileo hardware receiver for distress beacon application (industrial project with Kannad, Guidel, France);
- I developed a FPGA-based image acquisition/processing system for a CMOS sensor (industrial project with Siemens BT, Zurich).

## Professional skills

Main research and development activities:

- Global Navigation Satellites System (GPS, Galileo);
- images acquisition and processing systems;
- voice recognition.

Main expertise domains:

- Digital design conception, simulation, and synthesis;
- FPGA design implementation;
- signal processing algorithms development, simulation, and implementation.

## Professional education

Professional training:

- VHDL course for low-power consumption design 2006
- C++ introduction course 2004
- R&D projects management course 2004
- CMOS image sensors course 2003
- Omega price for best diploma work in electronics & physics 2002

**Diploma in electronics & physics with distinction** **1997-2002**
University of Neuchâtel

**Scientific baccalaureate** - Gymnase français de Biel/Bienne **1994-1997**

## Language skills

French: first mother tongue;
English: fluent (I speak, read and write English every day);
German: high school level (I speak German regularly).

## IT skills

Programming languages: Matlab, VHDL, C/C++, assembler;
Design conception softwares: Matlab (Mathworks), HDL Designer, Modelsim (Mentor Graphics), Quartus (Altera), Design Compiler (Synopsys);
Others: Windows based OS (Microsoft), Latex, Office (Microsoft).

## Hobbies

Music, saxophone (big-band Yellownote Bevilard), hi-fi, photograohy, Taebo, jogging, wines of Bordeaux.