Real-Time GNSS Software Receiver: Challenges, Status, and Perspectives

Marcel Baracchi-Frei, *University of Neuchâtel, Switzerland*
Grégoire Waelchli, Cyril Botteron, Pierre-André Farine, *Ecole Polytechnique Fédérale de Lausanne (EPFL), Electronic and Signal Processing Laboratory, Neuchâtel, Switzerland*

BIOGRAPHY

Marcel Baracchi-Frei received his degree of Physics-Electronics at the University of Neuchâtel in March 2003. In July 2003 he joined the group 'Electronics and Signal Processing Laboratory' (ESPLAB) of the University of Neuchâtel where he is currently working as PhD student. From July 2003 to December 2004 he was involved in a project in the wireless sensor network domain. He focused on microcontroller programming and designing, testing, and characterizing small, printable antennas. Since December 2004 he is working in the domain of GNSS receivers. He has over 3 years of research experience in embedded systems, where he implemented the software part of a GPS L2 receiver on an embedded microprocessor and focalized on the navigation part of a Galileo receiver. Currently he is working on a software GPS receiver project.

INTRODUCTION

The idea of a software receiver is to replace the data processing implemented in hardware with software and to sample the analog input signal as close to the antenna as possible. Thus, the hardware is reduced to the minimum (antenna and analog to digital converters) while all the signal processing is done in software. As current mobile devices (such as personal digital assistants and smartphones) include more and more computing power and system features it becomes possible to integrate a complete GNSS receiver with very few external components.

One advantage of a software receiver lies clearly in the low cost opportunity as the system resources such as the calculation power and system memory can be shared. Another advantage resides in the flexibility for adapting to new signals and frequencies. Indeed, an update can easily be performed by changing some parameters and algorithms in software while it would require a new re-development for a standard hardware receiver.

Updating capabilities may become even more important in the future as the world of satellite navigation is in complete effervescence: Europe is developing its own solution (named Galileo) that is foreseen to be operational in 2013; China is about to undertake a fundamental re-development of its current navigation system (named Compass); Russia is investing a huge amount of money in its GLONASS system to bring it back to full operation; and the U.S. GPS system will see some fundamental improvements during the next few years with new frequencies and new modulation techniques. At the same time, augmentation systems (either space based or land based) will be developed all over the world.

These future developments will increase the number of accessible satellites available to every user – with the advantage of better coverage and higher accuracy. However, to take full advantage of the new satellite constellations and signals, new GNSS receivers and algorithms must be developed.

In this paper, we will first give a short overview of the history of software receivers. Then, the term 'software receiver' will be explained and the different classes of software receivers presented. The third section is dedicated to the challenges that the development of software receivers is confronted with. This includes the high data rate and the demanding computational power. The next section provides an overview and a short description of the current status related to the algorithms of code and carrier generation, acquisition, tracking and baseband processing for software receivers. This also includes a discussion about Single Instruction Multiple Data (SIMD) operations and bit-wise (or vector) processing. Finally, the last section provides some concluding remarks and a research outlook.

HISTORY

During the 1990's, a U.S. Department of Defense (DoD) project named Speakeasy was undertaken with the objective of showing and proving the concept of a programmable waveform, multiband, multimode radio [1]. The Speakeasy project demonstrated the approach that underlies most software receivers: the analog to digital converter (ADC) is placed as near as possible to the antenna front-end, and all baseband functions that receive digitized intermediate frequency (IF) data input are processed in a programmable microprocessor using software techniques rather than hardware elements, such as correlators. The programmable implementation of all baseband functions offers a great flexibility that allows rapid changes and modifications. This property is an advantage in the fast changing environment of GNSS receivers as new radio frequency (RF) bands, modulation types, bandwidths, and spreading/dispreading and baseband algorithms are regularly introduced.

SOFTWARE REVEIVER: DEFINITION AND TYPES

The definition of a software receiver (SR) always brings some confusion among researchers and engineers in the field of communications and GNSS. For example, a receiver containing multiple hardware parts which can be reconfigured by setting a software flag or hardware pins of a chipset are regarded by some communication engineers to be a SR. In this paper, however, we will consider the widely accepted SR definition in the field of GNSS, that is, a receiver in which all the base-band signal processing is performed in software by a programmable microprocessor.

Nowadays, software receivers can be grouped in three main categories as shown in Figure 1.

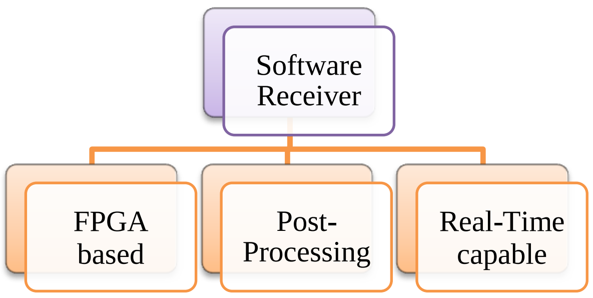

Figure 1 : Software receiver types

The first category regroups the receivers that are based on Field Programmable Gate Arrays (FPGAs), which are sometimes also referred to the domain of SR. These receivers can be reconfigured in the field by software.

The second category, post-processing receivers, includes, among others, the countless software tools or lines of code for testing new algorithms and for analyzing the GNSS signal, for example, to investigate GPS satellite failure or to decrypt unpublished codes.

Finally, the third category is the real-time capable software receivers group that will be further considered in this paper.

A modern GNSS receiver contains normally a RF front-end, a signal acquisition, a tracking, and a navigation block. A hardware-based receiver accomplishes the residual carrier removal, PRN code dispreading, and integration at the system sampling rate. Until the late 1990s, due to the limited processing power of microprocessors, these signal functions could only be practically implemented in hardware.

In 1990, researchers at the NASA/Caltech Jet Propulsion Laboratory introduced a signal acquisition technique for code division multiple access (CDMA) systems that was based on the Fast Fourier Transform (FFT) [2]. Since then, this method has been widely adopted in GNSS SR because of its simplicity and efficiency of processing load.

In 1996, researchers at Ohio University provided a direct digitization technique – called the bandpass sampling technique – that allowed the placing of ADCs closer to the RF portions of GNSS SRs. Until this time, the implemented SRs in university laboratories post-processed the data due to the lack of processing power mentioned earlier.

Finally, in 2001, researchers at Stanford University implemented a real-time processing-capable SR for the GPS L1 C/A signal [3].

However, the GNSS SR boom really started with the development of real-time processing capability. This was first accomplished on a digital signal processor (DSP) and later on a commercial conventional personal computer (PC). Today, the DSPs are more and more replaced by specialized processors for embedded applications.

CHALLENGES

The following chapter highlights some of the main challenges related to software receivers. This includes the problem of the high data rate when working with a nearly ideal implementation and also talks about the high processing power requirements for the base-band processing of the incoming GNSS signal.

Data rate

The ideal software receiver would place the ADC as close as possible to the antenna in order to reduce the hardware parts to the minimum (see Figure 2). In that sense, the most straightforward approach consists in digitizing the data directly at the antenna, without pre-filtering or pre-processing. But as the Nyquist theorem must be fulfilled (i.e. sampling with at least twice the highest signal frequency), this translates into a data rate that is, for the time being, too high to be processed by a microcontroller.

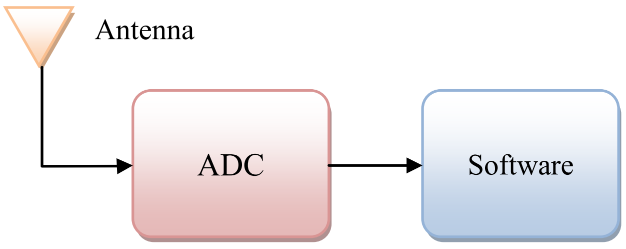

Figure 2 : Ideal software receiver

Considering the GPS L1 signal and assuming 1 quantization bit per sample, this leads to the following values:

$F_{GPSL1} = 1.575$ GHz
$F_{Sampling} \geq 2 * F_{GPSL1} = 3.15$ GHz
Data rate $\geq$ 3.15 GBit/s = 393 MB/s

In order to reduce the data throughput, a solution such as a low intermediate frequency (low IF) or a sub-sampling analog front-end must be chosen. In a low IF front-end, the incoming signal is down-converted to a lower intermediate frequency of several megahertz. This allows working with a sampling (and data) rate that can be more easily handled by a microcontroller.

The sub-sampling technique exploits the fact that the effective signal bandwidth in a GNSS signal is much lower than the carrier frequency. Therefore, not the carrier frequency but the signal bandwidth must be respected by the Nyquist theorem (assuming appropriate band-pass filtering). In this case, the modulated signal is under-sampled to achieve frequency translation via intentional aliasing. Again, if the GPS L1 signal is taken as an example with assuming 1 quantization bit per sample, this leads to the following values:

Bandwidth GPS L1 = 2 MHz
$F_{Sampling} \geq 2 *$ Bandwidth = 4 MHz
Data rate $\geq$ 4 MBit/s = 500 kB/s

However as the sub-sampling approach is still difficult to implement due to current hardware and resources limitations, a more classical solution based on an analog IF down-conversion is often chosen. That means that the signal is first down-converted to an intermediate frequency and afterwards digitized.

Base-band processing

Considering an IF based architecture, the ADC provides a data stream (real or complex) which is first shifted into base-band by at least one complex mixer. The signal is then multiplied with several code replicas (generally Early, Prompt, and Late) and finally accumulated. An example of a real data IF architecture is shown in Figure 3.

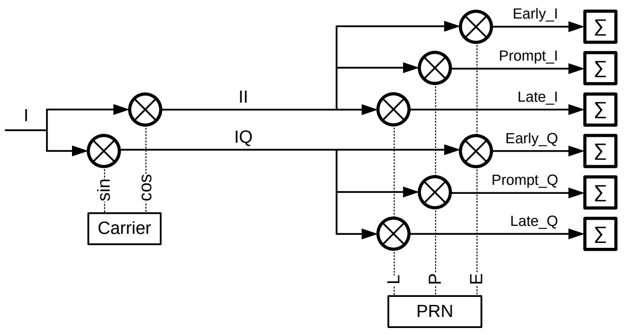

Figure 3 : Real IF architecture

In hardware receivers, the local code and carrier are generally generated in real-time by the means of a Numerically Controlled Oscillator (NCO) which performs the role of a digital waveform generator by incrementing an accumulator by a per-sample phase increment. The resulting value is then converted to the corresponding amplitude value in order to recreate the waveform at any desired phase offset. The frequency resolution is typically in the range of a few millihertz with a 32-bit accumulator and a sampling frequency in the range of a few megahertz.

Assuming that a look-up table (LUT) address can be obtained with 2 logical operations (one shift and one mask) and the corresponding LUT value read with 1 memory access - which is quite optimist - the amount of needed operations to generate the complex waveforms per channel becomes (see Table 1).

# integer additions	$3 * N_{Ch} * F_S * T_{Int}$
# interer multiplications	$4 * N_{Ch} * F_S * T_{Int}$
# logical operations	$3 * N_{Ch} * F_S * T_{Int}$

Table 1: Operations for code and carrier generation

The real-time carrier generation is computationally expensive and is consequently not suitable for a one to one software implementation. Former studies [4] demonstrated that, assuming that an integer operation and a multiplication take 1 and 14 CPU cycles, respectively (for an Intel Pentium 4 processor), the base-band operations (without carrier and code generation or navigation solution) would require at least a 3 GHz Intel Pentium 4 processor with 100% CPU load. Therefore, under these conditions, real-time operations are not suitable for embedded processors. Therefore standard hardware receiver architectures cannot be translated directly into software and consequently, new strategies must be developed to lower the processing load.

STATUS

A major problem with the software architecture is the important computing resources required for the base-band processing, especially for the accumulation process. As a straightforward transposition of traditional hardware based architectures into software would lead to an amount of operations which is not suitable for today's fastest computers, two main alternate strategies have been proposed in the literature: the first one relies on the utilization of Single Instruction Multiple Data (SIMD) operations which provide the capability of processing vectors of data. Since they operate on multiple integer values at the same time, SIMD could result in significant gains in execution speed for repetitive tasks such as base-band processing. However, SIMD operations are tied to specific processors and therefore severely limit the portability of the code. The second alternative consists in the bitwise parallel operations (sometimes also referred as vector processing in the literature), which exploit the native bitwise representation of the signal. The data bits are stored in separate vectors, one sign and one or several magnitude vectors, on which bitwise 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 is translated into a few simple parallel logical relations. While SIMD operations use advanced and specific optimization schemes, the latter methodology exploits universal CPU instructions set.

Single Instruction Multiple Data

In 1995, Intel introduced the first instance of Single Instruction Multiple Data (SIMD) under the name of Multi Media Extension (MMX). The SIMD are mathematical instructions that operate on vectors of data and perform integer arithmetic on eight 8-bit, four 16-bit, or two 32-bit integers packed into a MMX register (see Figure 4). On average, the SIMD operations take more clock cycles to execute than a traditional x86 operation. Anyhow, since they operate on multiple integers at the same time, MMX code can result in significant gains in execution speed for appropriately structured algorithms. Later SIMD extensions, SSE, SSE2, and SSE3, added eight 128-bit registers to the x86 instruction set. Additionally, SSE operations include SIMD floating point operations, and expand the type of integer operations available to the programmer.

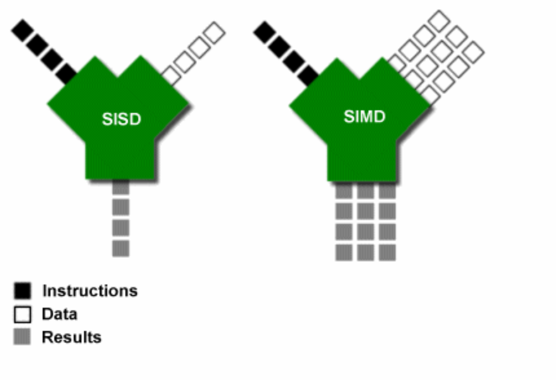

Figure 4 : Single Instruction Single Data vs. Single Instruction Multiple Data

SIMD operations are well fitted to parallelize the operations of the baseband processing (BBP) stage. In particular, they can be used to allow the PRN code mixing and the accumulation to be performed concurrently for all the code replicas. With the help of further optimizations such as instruction pipelining, more than 600% performance improvement with the SIMD operations compared to the standard integer operations can be observed [5]. For this reason, most of the software receivers with real-time processing capabilities use SIMD operations [4], [5], [6], [7].

Bitwise Operations (Vector Processing)

Bitwise operation (or vector processing) was first introduced in [8]. The method exploits the bit representation of the incoming signal where the data bits are stored in separate vectors on which bitwise parallel operations can be performed. Figure 5 shows a typical data storage scheme for vector processing.

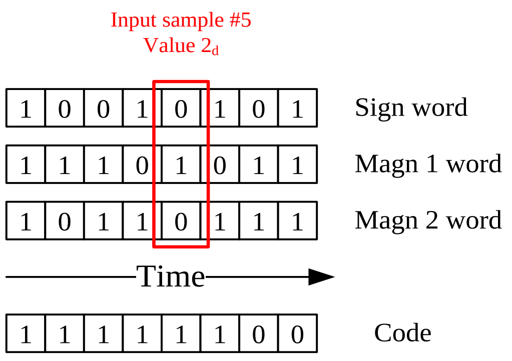

Figure 5 : Bitwise representation

The sign information is stored in the *sign word* while the remaining bit(s) representing the magnitude is (are) stored in the *magn word(s)*. The objective is to take advantage of the high parallelism and speed of the bitwise operations for which a single integer addition or multiplication is translated into simple parallel logical operations. The carrier mixing stage is reduced to one or a few simple logical operations which can be performed concurrently on several bits. In the same way, the PRN code removal only affects the sign word.

In [9] the complete code and carrier removal process requires two operations for each code replica (Early, Prompt, and Late). The complexity can be even further reduced by more than 30% by considering one single combination of early and late code replicas (typically early-minus-late). This way, the author claims an improvement of a factor 2 for the bitwise method compared to the standard integer operations.

The inherent drawback of this approach is the lack of flexibility: the complexity of the process becomes bit-depth dependent and the signal quantification cannot be easily changed (while performing BBP with integers allows the signal structure to change significantly without code modification).

To overcome this limitation, a combination of bitwise processing and distributed arithmetic can be used. This method was described in details in [10]. The power consuming operations are performed with bitwise operations and to be able to keep the flexibility of the calculations standard integer operations are used after the code and carrier removal. The passage between the two methods is done with the distributed arithmetic.

Code and carrier generation

Another important aspect in a software receiver is the code and carrier generation. As these tasks represent a huge processing load, new solutions have to be developed in this domain.

Code generation

The pseudorandom noise (PRN) codes transmitted by the satellites are deterministic sequences with noise-like properties that are typically generated with tapped linear feedback shift registers. But in order to save processing power, it is preferable for software applications to compute off-line the 32 codes and store them in memory.

A method that stores the different PRN codes in their oversampled representation (the code are pre-generated) was proposed in [8]. As the incoming signal code phase is random, the beginning of the first code chip is in general not aligned with the beginning of a word and may occur anywhere within it. To overcome this issue, either all the possible phases can be stored in memory or the code can be shifted appropriately during the tracking. While the first approach increases the memory requirements, the second requires further data processing in function of the phase mismatch. Regarding the Doppler compensation, all the PRN codes in the table are assumed to have zero Doppler shifts. The code phase errors due to this hypothesis are eliminated by choosing a replica code from the table whose midpoint occurs at the desired midpoint time. The only other effect of the zero Doppler shift assumption is a small correlation power loss which is not more than 0.014 dB if the magnitude of the true Doppler shift is less than 10 kHz [9]. This approach is very popular in the domain of software receiver and can be found in several solutions [4], [11], [12], [13], [14].

Carrier generation

The generation of a local carrier frequency is necessary to perform the Doppler removal. The standard trigonometric functions or the Taylor decompositions for the sinus and cosines computation are too heavy for a software implementation and are seldom considered.

However, several other techniques exist to reduce the computational load for the carrier generation: the values for the carrier can be pre-generated and then stored in lookup tables. This method was first introduced into a software receiver by [8]. As it would require several gigabytes of memory to store all the possible frequencies, the values are recorded on a coarse frequency grid with zero phases and at the RF front-end (over-)sampling frequency. The carrier will thus be available in an (over-)sampled version. The limited number of available carrier frequencies introduces a supplementary mismatch in the Doppler removal process. This error can be compensated with a simple phase rotation of the accumulation results. This method is very popular in the domain of software receivers and many solutions take advantage of it to avoid the power hungry real-time carrier generation [6], [7], [12].

Based on the same principle as above, [15] proposed a method that pre-computes a set of carrier frequency candidates to be stored in memory. The grid spacing is selected such as to minimize the loss due to Doppler frequency offset. Furthermore, in order to provide phase alignement capabilities of the carriers, a set of initial phases is also provided for each possible Doppler frequency, as illustrated in Figure 6.

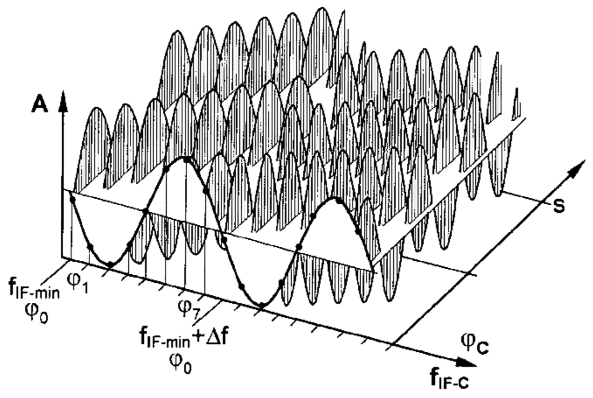

Figure 6 : Set of carrier frequency candidates

Contrarily to the approach in [8] and thanks to the phase alignement capabilities, the number of sampling points must not obligatorily correspond to an entire acquisition period. Therefore, the length of the frequency candidate vectors can be chosen with respect to the available memory space and becomes quasi independent of the sampling frequency.

Another approach consists in removing concurrently the Doppler from all received satellite signals [16]. The algorithm is implemented as a look-up table containing one single frequency and the carrier removal is performed for all channels with the same frequency, but the frequency error results normally in an unacceptable loss. To overcome this problem, the integration interval is split into sub-intervals for which a partial accumulation is computed. The result is rotated proportionally to the frequency mismatch in the same way as in the method described above. The algorithm can be applied recursively and with an appropriate selection of the sub-intervals, the total attenuation factor can be limited to a reasonable value. The author claims an improvement of up to 30% compared to the standard look-up table method with respect to the total complexity for both Doppler removal and correlation stages. Regarding the computational complexity, the Doppler removal stage remains unchanged with the difference that it is only performed once for all satellites. But the rotation needs to be done for each of the sub-intervals. However, this algorithm remains difficult to implement (number of samples varies in one or more full C/A code chip and the alignement of the data is different than the sub-interval boundaries).

Acquisition

During acquisition, the BBP unit shifts locally the code replica until it correlates with the incoming code. The receiver must also detect the satellites or space vehicles (SVs) present in the incoming signal by searching the possible Doppler frequencies. The acquisition thus consists in a two dimensional search process. Different acquisition techniques can be envisaged.

Serial Search

The serial search represents the classical approach and consists in sweeping the two dimensional code phase/Doppler spaces in a sequential manner. Every incoming sample is multiplied with the local carrier replica and a correlation peak is searched sequentially, as illustrated in Figure 7.

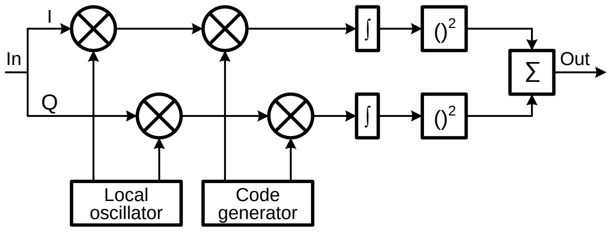

Figure 7 : Serial search architecture

Parallel code search

The parallel code search tests all the code phases in parallel for a given Doppler frequency. The input signal is transformed into the frequency domain via a Discrete Fourier Transform (DFT). The DFT of the locally generated PRN code is also computed. After multiplication of these two sets of coefficients, the inverse DFT is performed to determine if a correlation peak is present. If not, the operation is repeated for the next Doppler frequency. The process is illustrated in Figure 8. As compared to the previous method, the parallel code phase search method reduces the search space to the different carrier frequencies. As the Fourier Transform of the replica PRN code can be pre-computed and stored, each of the bin searching consists in performing one Fourier Transform and one Inverse Fourier Transform.

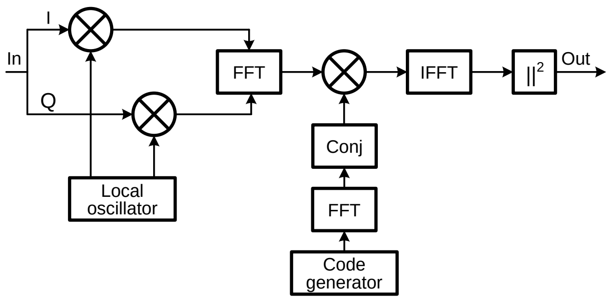

Figure 8 : Parallel code search architecture

Parallel frequency search

The parallel frequency search looks for the peak in the frequency domain by testing all Doppler bins at once and all code phases individually. The baseband signal is multiplied with the locally generated PRN code in order to form P consecutive partial correlations with a pre-detection time T_C which is P times smaller than the integration time. The P results are regrouped into a vector on which a N-point FFT is computed. If no correlation peak is detected, the operation is repeated with the next code phase, as shown in Figure 9.

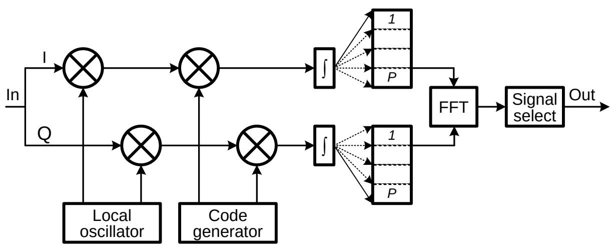

Figure 9 : Parallel frequency search architecture

AVAILABLE SOFTWARE RECEIVERS

Today, software receivers can be found at either university or commercial level. The development not only includes programming solution but also the realization of dedicated RF front-ends. As these RF front-ends are able to capture more and more frequencies with increasing bit-rates, the PC-based software receivers require a comparably complex interface to transfer the digitized IF samples into the computer's memory.

Two classes of PC-based GNSS SR front-end solutions can be found. The first one uses commercially available ADCs that are either connected directly to the PC (for example, via the PCI bus) or that are working as stand-alone devices. The ADC directly digitizes the received IF signal, which is taken from a pure analog front-end. This solution is often found at the university and research institute levels where a high amount of flexibility is required. As an example the Department of Geomatics Engineering of the University of Calgary [17], the Cornell University [18], and the University FAF Munich's Institute of Geodesy and Navigation [19] shall be mentioned here.

The second solution is based on front-ends that integrate an ADC plus an USB 2.0 interface. Currently, a quite impressive number of commercial and R&D front-ends are available for the GNSS market. NordNav (bought by CSR) [20] and Accord [21] were among the first to provide USB-based solutions. Another interesting development comes from the University of Colorado, which in an OpenGPS forum published all details on the RF and USB section. More and more companies announced and still announce front-ends that are not only capable of capturing a single frequency, but several different bands. To be able to deal with this increasing bandwidth, the USB port is very well suited for SR development and its maximum theoretical transfer rate of 480 MBit/s allows realizing GPS/Galileo multi-frequency high bandwidth frontends. The USB approach is one of the most important cornerstones of SR development.

Software receivers for the embedded market

As mentioned in the introduction, the embedded market will become important during the next years. A growing number of receivers are developed for this market, supporting different embedded platforms (e.g. Intel XScale, ARM based, and DSP-based). Several companies offer already commercial software receivers for the embedded market, among others NordNav (bought by CSR) [20], SiRF [22], ALK Technologies Inc. [23], and CellGuide [24].

Commercial PC-based receivers

The first commercial GPS/Galileo receiver for a PC platform was presented in 2001 by NordNav (bought by CSR) [20]. This SR can be compared to a normal GPS receiver, although the CPU load of this solution is still quite impressive.

Several other solutions are presented lately. One of the first (car) navigation solution was presented by ALK Technologies [23] under the name CoPilot. The CPU load was drastically reduced and this solution works on a standard commercial personal computer. The client does not really see a difference compared to a solution that is based on a hardware receiver.

Software receivers for research activities

The use in teaching and for training is one of the most valuable and obvious application for software GNSS receivers. Receivers, for which the source code is available, allow the observation and inspection of almost every signal data by the researcher.

Several textbooks have been published related to software GNSS receivers. The pioneer in this area is James Bao-yen Tsui who wrote the first book on software receivers in 2000 with the title *Fundamentals of Global Positioning System Receivers: A Software Approach* (Wiley-Interscience, updated in 2004). Kai Borre et al. published in 2006 a book that comes with a complete (post-processing) software receiver written in Matlab: *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach* (Birkhäuser Boston, 1st edition).

The European Union is financing the development of receivers for the upcoming Galileo system. One of the projects was the Galileo Receiver Analysis and Design Application (GRANADA) simulation tool. Running under Matlab, GRANADA is realized as a modular and configurable tool with a dual role: test-bench for integration and evaluation of receiver technologies, and SR as asset for GNSS application developers.

Other companies provide toolboxes (in Matlab or C) that allow testing of new algorithms in a working environment and inspecting almost all data signals. The solutions from Data Fusion Corporation (DFC) [25] and NAVSYS [26] shall be mentioned here.

OUTLOOK

Software receivers have found their place in the field of algorithm prototyping and testing for a long time. Nowadays they also play a key role for certain special applications. What remains unclear today is if they will enter and change drastically the embedded market or succeed as generic high-end receivers.

A software GNSS receiver offers different advantages including design flexibility, faster adaptability, faster time-to-market, higher portability and easy optimization at any algorithm stage. However, a major drawback persists in the slow throughput and the high CPU load.

Many different companies and universities have projects running that aim at optimizing and developing new algorithms and methods for a software implementation. The development not only includes the software level, but also enlarges in the direction of using additional hardware that is already available on a standard PC (for example, using the high performance graphic processing unit (GPU) for calculating the local carrier [27]).

On the opposite end of the spectrum from the mass market, the following factors seem to ensure that, sooner or later, high-end software receivers will be available:

- High bandwidth signals (GPS and Galileo) can already be transferred into the PC in real-time and processed.
- The processing power is increasing allowing real-time processing with a limited amount of multi-correlators. The introduction of new multi core processors will be advantageously for software receivers.
- Post-processing is one of the most important benefits of a software receiver as it allows a re-analysis of the signal several times with all possible processing options. The increasing hard disk capacity allows the storage of several long data sequences.
- Some signal processing algorithms are much easier to implement in software than in hardware (such as frequency domain tracking or maximum likelihood tracking). Those methods require complex operations at the signal level.

REFERENCES

[1] R. Lackey und D. Upmal, “Speakeasy: The Military Software Radio,” *IEEE Communications Magazine*, S. 56-61.

[2] D. van Nee und A. Coenen, “New Fast GPS Code Acquisition Technique Using FFT,” *Electronics Letters*, vol. 27, Jan. 1991, S. 158-160.

[3] D.M. Akos, P. Normark, P. Enge, A. Hansson, und A. Rosenlind, “Real-Time GPS Software Radio Receiver,” Long Beach, CA: 2001.

[4] G.W. Heckler und J.L. Garrison, “Architecture of a Reconfigurable Software Receiver,” Long Beach, CA: 2004.

[5] G.W. Heckler und J.L. Garrison, “SIMD correlator library for GNSS software receiver,” *GPS Solutions*, 2006.

[6] T. Pany, S.W. Moon, M. Irsigler, B. Eissfeller, und K. Fürlinger, “Performance assessment of an under sampling SWC receiver for simulated high bandwidth GPS/Galileo signals and real signals,” Portland, OR: 2003.

[7] S. Charkhandeh, M. Petovello, R. Watson, und G. Lachapelle, “Implemenation and Testing of a Real-Time Software-Based GPS Receiver for x86 Processors,” Monterey, California: 2006.

[8] B. Ledvina, S. Powell, und P. Kintner, “A 12-Channel Real-Time GPS L1 Software Receiver,” 2002.

[9] B. Ledvina, M. Psiaki, S. Powell, und P. Kintner, “Real-time software receiver,” U.S. Patent US0227856.

[10] G. Waelchli, M. Baracchi-Frei, C. Botteron, und P. Farine, “Performances of a new

correlation algorithm for a platform-independent GPS software receiver," Anaheim CA: 2009.

[11] Y. Chen und J. Juang, "A GNSS Software Receiver Approach for the Processing of Intermittent Data," 2007.

[12] A. Fridmann und S. Semenov, "Architectures of Software GPS Receivers," *GPS Solutions*, vol. 3, 2000, S. 58-64.

[13] M.L. Psiaki, "Real-Time Generation of Bit-Wise Parallel Representation of Over-Sampled PRN Code," *IEEE Trans. on Wireless Communication,* vol. 5, März. 2006.

[14] J. Tian, Q. HongLei, Z. JunJie, und L. Yang, "Real-time GPS Software Receiver Correlator Design," 2007.

[15] P. Normark und C. Stahlberg, "Spread spectrum signal processing," U.S. Patent WO 2004/036238.

[16] M. Petovello und G. Lachapelle, "An Efficient New Method for Doppler Removal and Correlation with Application to Software-Based GNSS Receivers," Fort Worth, TX: 2006.

[17] http://www.geomatics.ucalgary.ca

[18] http://www.cornell.edu

[19] http://ifen.bauv.unibw-muenchen.de

[20] http://www.csr.com

[21] http://www.accord-products.com

[22] http://www.sirf.com

[23] http://www.alk.eu.com

[24] http://www.cell-guide.com

[25] http://www.datafusion.com

[26] http://www.navsys.com

[27] M. Petovello et al., "Architecture and Benefits of an Advanced GNSS Software Receiver", *International Symposium on GPS/GNSS 2008, Tokio*, 2008.