A Real-Time Software Receiver for the GPS and Galileo L1 Signals

B. M. Ledvina,
Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University
M. L. Psiaki, T. E. Humphreys,
Sibley School of Mechanical and Aerospace Engineering, Cornell University
S. P. Powell, and P. M. Kintner Jr.,
School of Electrical and Computer Engineering, Cornell University

BIOGRAPHY

Brent M. Ledvina will be an Assistant Professor in the Electrical and Computer Engineering Department at Virginia Tech in Spring 2007. He received a B.S. in Electrical and Computer Engineering from the University of Wisconsin at Madison and a Ph.D. in Electrical and Computer Engineering from Cornell University. His research interests are in the areas of ionospheric physics, space weather, estimation and filtering, and GPS/GNSS technology and applications.

Mark L. Psiaki is a Professor of Mechanical and Aerospace Engineering at Cornell University. He received a B.A. in Physics and M.A. and Ph.D. degrees in Mechanical and Aerospace Engineering from Princeton University. His research interests are in the areas of estimation and filtering, spacecraft attitude and orbit determination, and GNSS technology and applications.

Todd E. Humphreys is a graduate student in the Sibley School of Mechanical and Aerospace Engineering. He received his B.S. and M.S. in Electrical and Computer Engineering from Utah State University. His research interests are in estimation and filtering, spacecraft attitude determination, GNSS technology, and GNSS-based study of the ionosphere and neutral atmosphere.

Steven P. Powell is a Senior Engineer with the Space Plasma Physics Group in the School of Electrical and Computer Engineering at Cornell University. He has M.S. and B.S. degrees in Electrical Engineering from Cornell University. He has been involved with the design, fabrication, and testing of several GNSS receivers.

Paul M. Kintner, Jr. is a Professor of Electrical and Computer Engineering at Cornell University. He received a B.S. in Physics from the University of Rochester and a Ph.D. in Physics from the University of Minnesota. His research interests include the electrical properties of upper atmospheres, space weather, and developing GNSS instruments for space science. He is a Fellow of the APS.

ABSTRACT

A real-time interoperable GPS and Galileo L1 software receiver has been developed and tested. The receiver has 12 channels for tracking GPS satellites and 12 channels for tracking Galileo satellites. The receiver consists of a GPS L1 RF front end, data parallelizing and acquisition hardware, and software routines running on a 3.4-GHz Pentium processor. The software is composed of bit-wise parallel correlation routines, code and carrier tracking loops, data demodulation routines, and navigation solution code.

The main contributions of this work are the demonstration of a GPS L1 C/A code RF front end being used with the Galileo L1 binary-offset-carrier(1,1) (BOC(1,1)) signals and the demonstration of an interoperable GPS and Galileo real-time software receiver. This RF front end uses a commercial off-the-shelf (COTS) GPS RF front end designed for the L1 C/A code signal. To accommodate the Galileo L1-B BOC(1,1) signal, modifications to the delay-locked-loop (DLL) were required. The downside to the approach is that this particular RF front end induces a 4.6 dB loss in carrier-to-noise in the Galileo signals, because of the narrow filtering intended for the narrower GPS L1 C/A code signal. The successful live tracking of GPS and Galileo satellites using this RF front end demonstrates the feasibility of using RF front ends designed for the GPS L1 C/A code signals in interoperable GPS and Galileo receivers or Galileo receivers.

The GPS and Galileo L1 software receiver tracks 24 channels in real time. It requires 46% of the processing capabilities of a 3.4 GHz Intel Pentium 4 PC.

I. INTRODUCTION

A real-time global navigation satellite system (GNSS) receiver provides distinct advantages in an evolving signal and code environment. The current Global Positioning System (GPS) has began the process of modernization to expand its capabilities to include new civilian codes on the L2 frequency, and evenutally on the L5 frequency. Already, two GPS Block IIR-M satellites are in orbit and broadcasting the civilian L2 signal. In addition, the European Union's Galileo satellite navigation system has launched its first test satellite, GIOVE-A, nearly nine months ago. In the near term, a receiver designed will have to weigh the pros and cons of developing complex hardware correlator chips to take advantage of the rapidly-evolving signal environment. Since certain users will want access to these signals immediately and others will be willing to wait until a full constellation is available to provide the maximum performance benefit, the trade-offs from receiver design and marketing perspective are, to say the least, bewildering. A software receiver can utilize the new signals without the need for a new correlator chip. Given a suitable RF front end, new frequencies and new pseudo-random number (PRN) codes can be used simply by making software changes. Thus, software receiver technology will lessen the risks involved for designers during the period of transition to the new signals. It is clear that a clear advantage can gained from using a software radio approach to design Global Navigation Satellite System (GNSS) receivers.

This work focuses on the evaluation and testing of a GPS and Galileo L1 RF front end and the testing of a GPS and Galileo software receiver that utilizes the GPS L1 C/A code and the Galileo L1-B BOC(1,1) code. The GIOVE-A L1-B PRN code was obtained from [1]. The Galileo L1-C BOC(1,1) code is not used in this software receiver, but it is straightforward to add functionality to acquire and track it.

Current GPS and Galileo hardware receivers are hard to come by, primarily due to the lack of Galileo satellites in orbit and the uncertainty about the Galileo Interface Control Document (ICD) and assoicated licensing fees. Regardless of these issues, it is instructive to contrast hypothetical GPS and Galileo hardware receivers with a software receiver. A hardware GPS and Galileo receiver can be broken down into various components. First, an L1 antenna, possibly followed by a pre-amp, receives the L-band GPS and Galileo signals. After the antenna comes an RF section that filters and down converts the GHz signals to intermediate frequencies in the MHz range. The RF section also digitizes the signal. The next section contains a correlator chip that separates the signal into different channels allocated to each satellite. A modern receiver has 12 for GPS signals and 12 channels for Galileo signals. For each satellite and each carrier frequency, the correlator mixes the Doppler-shifted intermediate frequency signal to baseband and correlates it with a local copy of a PRN code. The final components of the receiver consist of software routines that track the signals by controlling carrier and code numerically-controlled oscillators in the correlator chip, that decode the navigation messages, and that compute the navigation solution.

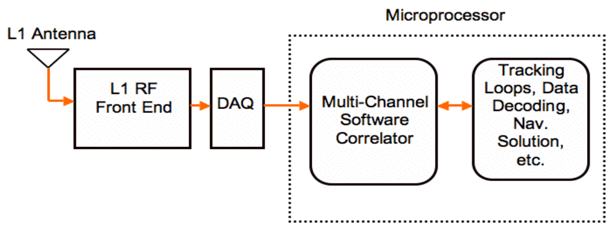

Microprocessor
L1 Antenna
L1 RF Front End
DAQ
Multi-Channel Software Correlator
Tracking Loops, Data Decoding, Nav. Solution, etc.

Fig. 1. A GPS and Galileo software receiver showing the separation between special-purpose hardware and general-purpose hardware.

A software receiver differs from a hardware receiver in one important way. The functions of the correlator chip are moved to software that runs on a general-purpose microprocessor. This processor could be a Pentium 4 processor in a personal computer or a digital signal processor (DSP). This move is illustrated in Fig. 1, where the software correlator is shown running on a general-purpose microprocessor. The RF front end outputs a binary bit stream. A data buffering and acquisition system reads the bit stream into the microprocessor's memory. The bit stream is then processed by the software correlator. The multi-channel correlator shown in Fig. 1 processes inputs from a bit stream containing GPS and Galileo L1 signals.

GPS software receivers have received a lot of interest recently [2], [3], [4], [5], [6], [7]. The work presented in this paper improves upon the software receiver of [6]. The receiver described here uses an analog mixing GPS L1 RF front end and has been tested using live GPS satellite signals and the first Galileo test satellite, GIOVE-A.

The remaining portions of this paper review the internal workings of a 24-channel real-time interoperable GPS and Galileo software receiver, describe the GPS RF front end, and present experimental performance results for this system. The second section provides an overview of the components of the software receiver. Section III describes the system configuration including the data acquisition system and the use of previously-existing GPS software for tracking and navigation. Section IV describes the evaluation and testing of a GPS and Galileo L1 RF front end. Section V reviews the bit-wise parallel software correlation technique of [4] and [5]. Section VI reviews the real-time generation of over-sampled PRN codes in [6] and [7]. Section VII discusses computational and memory requirements of the software correlator. Section VIII presents receiver performance results. The last section gives a summary and concluding remarks.

II. OVERVIEW OF THE SOFTWARE RECEIVER COMPONENTS

The software receiver described in this paper is composed of both hardware and software components. The RF front end is described in detail, but the software algorithms are simply reviewed as they have been covered in previous works.

Fig. 1 illustrates the general functionality of the software receiver. A commercial off-the-shelf (COTS) GPS L1 C/A code RF front has been re-used to provide a common RF signal path for both the GPS and Galileo L1 signals. The data buffering and acquisition hardware transfers the digital data stream into the PC for processing. One important aspect of the data buffering hardware is that it bit packs RF samples into shift registers before the data is read into the PC. This both reduces the bandwidth needed to read the data into the PC and sets up the software correlation process.

The remainder of the software receiver consists of software running on the 3.4 GHz PC. The digital data stream output from the RF front end is processed by the software correlator. This correlator processes 24 or more channels of the GPS L1 C/A code signals and Galileo L1-B BOC(1,1) code signals in real time. The correlator uses two technologies in order to process the RF front end data in an efficient manner. Without these two technologies, it would be reasonably challenging to build a real-time 24-channel GPS and Galileo software receiver using current microprocessors. The first technology, called bit-wise parallel signal processing, is a method for processing 32–64 RF samples in parallel. This method requires that the RF front end samples are bit packed into words prior to being read into the PC. Base-band mixing, code mixing, and accumulation are performed using bit-wise parallel processing of these bit-packed words. This speeds up the processing in the correlator by a factor of 2–4 as compared to using fixed-point multiply-and-accumulate (MAC) operations [4].

The second technology is a method for efficient real-time generation of over-sampled bit-packed PRN codes. This technology is helpful since Galileo L1 signals have longer periods than the L1 C/A code signals. In software receivers, it is computationally advantageous to pre-compute and store PRN code replicas, but the relatively long length of Galileo L1-B and L1-C BOC(1,1) PRN codes makes this approach less attractive, due to the required memory for storing the code replicas. Conserving memory can be particularly important on embedded platforms. The downside to this technology is that it adds a significant computational cost to the software receiver, increasing the required computational cost by roughly 40% [7]. Note that it is not necessary to use this second technology, unless fast memory is in short supply.

The aspect of real-time generation of over-sampled bit-packed PRN codes that is new here is how to deal with the BOC(1,1) signals. This signals effectively increases the nominal L1-B chipping rate by a factor of two. This induces an unreasonable increase in the memory required for the real-time generation of over-sampled Galileo L1 BOC(1,1) codes. A fix-up for the algorithm described in [7] is included in Appendix A.

The remaining components of the receiver are software for tracking the GPS and Galileo satellite signals, decoding the 50 bps GPS navigation message, computing the GPS navigation solution, displaying the receiver's operation, interacting with the user, etc. Many of these components were available using previously-existing GPS software. Some effort was needed to adapt these algorithms to the Galileo L1 signals. Note that the GIOVE-A's navigation message does not conform to the Galileo ICD, thus the functionality for decoding and interpreting the Galileo navigation message have not been included in this receiver.

III. SYSTEM CONFIGURATION

A software receiver relies on a general-purpose microprocessor to handle the signal processing of the correlator, tracking algorithms, navigation solution, etc. The current system uses a personal computer with a 3.4 GHz Intel Pentium 4 processor running the Real Time Application Interface (RTAI) (http://www.rtai.org) operating system. RTAI is a hard real time variant of Linux implemented as a set of patches to the standard Linux kernel. Due to its real-time optimized design, RTAI provides low-latency interrupt responsiveness along with the ability to execute threads at regular intervals. This translates into a highly efficient and responsive operating system that reliably executes time critical code. An additional feature of RTAI is that it retains the functionality of Linux by running the kernel as the lowest priority thread. Thus, it is surprisingly easy to develop, test, debug, and run real-time software.

The GPS L1 RF front end, which is described in the next section, outputs a digital data stream that is input into a data acquisition system. The data acquisition system reads the pair of digitized sign and magnitude bits from the RF front end into the PC. To make the process of reading data into the PC more efficient and to prepare for efficient correlation calculations, the DAQ card reads 32 bits of buffered samples at a time. A series of shift registers buffer the data, packing the L1 front end sign and magnitude bits into a 32-bit word. A divide-by-16 counter converts the 5.714 MHz clock down to 357.14 KHz, which provides a signal indicating when the buffer is full.

The data acquisition system consists of a PC card and driver software. The card is a National Instruments PCI-DIO-32HS (6533) digital I/O card. Important features of this card are its 32 digital input lines, its direct memory access (DMA), and the availability of a driver for RTAI. The driver comes from a suite of open source drivers and application interface software for DAQ cards known as COMEDI (COntrol and MEasurement and Device Interface), which is freely available.

Use of Existing Receiver Software

Existing receiver software is important to the implementation of this real-time GPS software receiver. The Mitel GPSArchitect GPS L1 C/A code receiver was ported to RT-Linux [8], subsequently ported to RTAI, and herein is referred to as Cascade. Since Cascade provides standard GPS functions such as signal tracking, data demodulation, and computation of the navigation solution, it is included as part of the real-time software receiver.

Two new developments to the Cascade software were required for it to operate with the new Galileo signals. First, a set of code and carrier tracking loops for the Galileo L1-B BOC(1,1) signals have been developed. The Cascade software already has a delay-locked loop (DLL) for code tracking and both a frequency-locked loop (FLL) and a phase-locked-loop for (PLL) for carrier tracking. These loops are for tracking the L1 C/A code signal and GPS L2 CM/CL code signals. Modified versions of these loops were implemented for the Galileo L1-B BOC(1,1) signals. The most important modifications involved changing the DLL discriminator and code phase error estimate to deal with the band-limited BOC(1,1) code and increasing the accumulation interval to 4 ms. Adjusting the PLL and FLL was straightforward, and [9] can be used as a reference. The second main development was adding an FFT-based acquisition routine.

The modifications to the DLL were required due to the Galileo BOC(1,1) signal's new early-minus-late correlation function and the band-limiting filter in the GPS L1 RF front end. The code phase error can be computed:

$$e_{codek} = \left[\frac{1}{|\frac{dR_{eml}}{d\tau}|_0}\right]\left[\frac{I_{emlk}I_{pk} + Q_{emlk}Q_{pk}}{I_{pk}^2 + Q_{pk}^2}\right] \quad (1)$$

where I_{pk} and Q_{pk} are the prompt accumulations for the accumulation interval t_k to t_{k+1}, I_{emlk} and Q_{emlk} are the early-minus-late accumulations, and $|dR_{eml}/d\tau|_0$ is the absolute value of the slope of the early-minus-late cross correlation function evaluated at $\tau = 0$ chips.

A simple carrier-aided discriminator has the form:

$$t_{k+2} = t_{k+1} + \frac{0.004}{1 + \frac{\omega_{k+1}}{2\pi \times 1575.42 \times 10^6}} - H\frac{e_{codek}}{1.023 \times 10^6} \quad (2)$$

where ω_{k+1} is the carrier PLL's Doppler shift estimate in rad/sec and the value $H = 0.025$ gives a 1-Hz DLL bandwidth.

The correct value for $|dR_{eml}/d\tau|_0$ is a function of the bandwidth of the RF front end and of the time offset between the early and late codes. The narrowest filter in the RF front end used with this receiver has a 1-dB pass-band of 1.9 MHz [10]. Choosing a 0.3 chip early-minus-late spacing for the 2.046×10^6 chips/sec L1-B BOC(1,1) code with this filter bandwidth gives a slope magnitude of the early-minus-late function of approximately 4.5970 per chip.

Second, an FFT-based acquisition method for detecting the Galileo L1-B BOC(1,1) signal has been developed. There are two compelling reasons why FFT-based acquisition is needed for the Galileo BOC(1,1) signals. First, the signals have 4 ms code periods, which makes brute-force time-domain acquisition take four times as long as that for the GPS L1 C/A code. Second, the BOC(1,1) signal in code-phase space contains a main lobe and two side loads that nominally have 25% as much power as the main lobe. In brute-force time-domain acquisition, where the correlator is linearly progressing through different code phase offsets, it would be easy to mistake one of the side lobes as the main lobe. This mistake would cause a ranging and carrier phase bias that would perturb the navigation solution.

FFT-based acquisition can take a significantly long time to complete for a full set of Doppler shift offsets and code phase offsets. If it takes too long, it is difficult to extrapolate the code start times forward using the estimated code chipping rate from the carrier Doppler shift rate. Because of this time concern, a revised approach has been used. For each channel, a full FFT-based acquisition is performed. This can take up to 10s of seconds on a Pentium 4 when 12 channels are attempting acquisition simultaneously. After each channel completes the full FFT-based acquisition, an FFT-based acquisition is performed again, but only at the Doppler shift corresponding to the peak $I^2 + Q^2$ acquisition statistic output from the previous full acquisition. This provides a reliable way to perform a cold start for the receiver. If almanac or ephemeris data is available, along with a reasonable estimate of the receiver clock drift rate, this approach can be simplified.

IV. USE OF A COTS GPS RF FRONT-END

One of the design goals of this project has been to re-use a GPS L1 C/A-code RF front-end for receiving the Galileo L1 BOC(1,1) signal. The COTS front-end that has been chosen is the Zarlink/Plessey GP2015 [10]. This front-end has been chosen because the project already had several of them along with the necessary hardware interfaces and software to stream their outputs to a computer. An advantage of this RF front-end is that its sampling rate is only 5.714 MHz. This relatively low sampling rate yields a manageable computational burden for the real-time software receiver's software correlator. The challenge associated with using this RF front-end is its narrow bandwidth: The 1 dB bandwidth of its narrowest filter is 1.9 MHz. This narrow bandwidth is needed in order to avoid aliasing of out-of-band noise when sampling at 5.714 MHz. It is sufficient to pass all of the main lobe of the GPS L1 C/A code.

This bandwidth is not wide enough to pass all of the Galileo L1 BOC(1,1) signal's two main lobes. Several different options have been tried in order to determine how best to use this narrow bandwidth to pass a significant part of the BOC(1,1) signal. Each of these options has two design goals: to minimize the loss of carrier-to-noise ratio (C/N_0) that is caused by the loss of out-of-band parts of the signal and to minimize distortion of the BOC(1,1) signal. Distortion can cause additional processing losses of C/N_0, and it can complicate the development of discriminators for the code-tracking delay-lock loop (DLL).

The three different design options that have been considered are illustrated in Fig. 2. The design problem at hand can be understood by considering the signal spectra that are illustrated by the small power spectrum plot at the center of the left-hand side of the figure. The blue part of the plot is the main lobe of the L1 C/A-code, and the red parts are the two main lobes of the BOC(1,1) signal. The 1.9 MHz pass band of the narrowest filter of the GP2015 is depicted by the black curves on the figure's other three power spectrum plots, which are the plots associated with outputs of GP2015 RF front-ends. The pass band is wide enough to accommodate the entire C/A code main lobe, but not wide enough to fully accommodate both BOC(1,1) lobes. Thus, the problem of passing the BOC(1,1) signal through this RF front-end can be compared to the problem of trying to pass a Texas football player through a Volkswagen Beetle: he is too big to fit.

The three different solutions to the RF front-end design problem are shown in Fig. 2 separated by the horizontal dash-dotted lines. The top-most design is the first

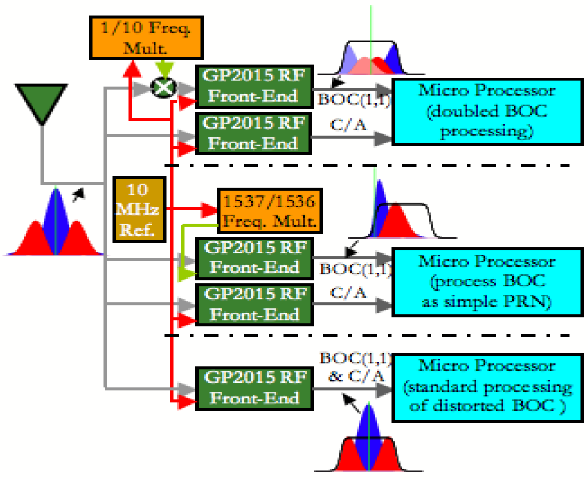
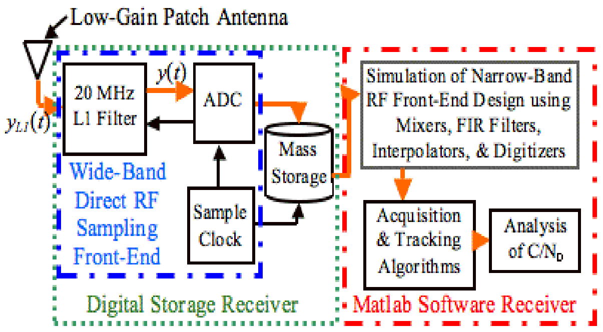


Fig. 2. Three design options for using a narrow-band L1 C/A-code RF front-end to receive Galileo L1 BOC(1,1) signals.

approach that has been considered. It uses two separate GP2015 RF front-ends. The lower of the two is used without modification in order to receive the GPS L1 C/A code. The upper front-end processes a signal that has first been mixed with a 1 MHz sine wave. This mixing signal is generated from the front-ends' 10 MHz reference oscillator by using a frequency divider. This mixing signal translates the center of the upper lobe of the BOC(1,1) signal to a point just below the L1 frequency, and it translates the lower lobe's center to a point just above the L1 frequency. This result is depicted by the power spectrum plot that is attached to the output of the upper front-end. Each such spectrum plot depicts the intermediate value of the nominal L1 frequency as a green vertical line and the pass band of the front-end's narrowest filter as a black curve The two BOC(1,1) lobes' heights have been halved because half of the power in each lobe gets translated 1 MHz in the wrong direction, which places it outside of the filter's pass band. Note that the blue C/A code lobe gets split into two lobes by this mixing process, and the filter attenuates out-of-band portions of each of these lobes.

The effectiveness of the top RF front-end design strategy has been analyzed by using the experimental set-up depicted in Fig. 3. This set-up uses a wide-band RF front-end that is based on direct RF sampling in order to receive, digitize, and record a signal with a 20 MHz bandwidth centered at the L1 frequency. Next, this signal is processed off-line in a Matlab simulation that uses mixers, finite-impulse response (FIR) filters, down-sampling via interpolation, and low-bit digitization in order to simulate the processing and outputs of any given design for a narrow-band RF front-end and associated analog hardware.



Low-Gain Patch Antenna
$y_{L1}(t)$
20 MHz L1 Filter
$y(t)$
ADC
Mass Storage
Sample Clock
Wide-Band Direct RF Sampling Front-End
Digital Storage Receiver
Simulation of Narrow-Band RF Front-End Design using Mixers, FIR Filters, Interpolators, & Digitizers
Acquisition & Tracking Algorithms
Analysis of C/N0
Matlab Software Receiver

Fig. 3. Evaluation test bed for RF front-end design options.

It was originally thought that the upper front-end design of Fig. 2 would be an effective means of receiving the BOC(1,1) signal by virtue of its strategy of squeezing both main lobes through the filter's 1.9 MHz pass band. Simulation and analysis of this design has shown that it is feasible, but less effective than had been anticipated. This design causes a loss of 4.6 dB of C/N_0 in comparison to a wide-band RF front-end. 3 dB of the loss comes from the fact that noise from the upper lobe gets mapped onto the lower lobe and vice versa. This noise mapping effectively doubles the noise density without doubling the signal power. A loss of 0.7 dB occurs because of the loss of side-lobe power, and a loss of 0.9 dB occurs because of signal distortion; the receiver's signal replica does not exactly match the filtered version of the signal. These latter contributions to the net loss have been determined by an FFT-based analysis of the BOC(1,1) signal's passage through a simple model of the GP2015 RF front-end that includes a "brick-wall" filter. In addition to these C/N_0 losses, this design has the negative feature that it places double the processing burden on the receiver's software correlator because the correlator must process separate accumulations for the upper and lower lobes of the signal.

The disappointing performance of the top RF front-end led directly to consideration of the middle design of Fig. 2. Like the top design, its lower GP2015 front-end is unmodified, and it receives the C/A code. The upper GP2015 in the middle design is used to receive only the upper lobe of the BOC(1,1) signal, as depicted in the power spectrum that is attached to the output of this front-end. The upper lobe of the BOC(1,1) signal gets mapped approximately to the center of the filter pass band by modification of the reference oscillator input to the GP2015: the 10 MHz reference signal is multiplied by 1537/1536 in a frequency synthesizer before input to the front-end.

The middle RF front-end design experiences BOC(1,1) C/N_0 losses that are comparable to those of the upper design. The loss of the lower lobe and the loss of out-of-band parts of the upper lobe add up to 4.2 dB. Signal distortion mismatch between the received code and the receiver replica adds an additional 0.4 dB of loss to yield a total loss of 4.6 dB. The same simulation and analysis techniques have been used to determine these losses as have been used for the upper RF front-end design.

Despite having comparable C/N_0 losses, this second BOC(1,1) front-end design is superior to the first design. Its superiority comes from the simplified signal processing that it allows in the real-time software receiver. Only one set of correlations are required in order to deal with the single lobe that passes through the RF front-end. These correlations are computed using the pseudo-random number (PRN) code part of the BOC(1,1) signal as though it were mixed with a carrier signal using simple bi-polar phase-shift keying (BPSK). The only impact of the binary-offset carrier (BOC) signal is a shift of the nominal carrier frequency to 1575.42 + 1.023 MHz.

The disappointing C/N_0 losses in the upper two RF front-end designs of Fig. 2 led naturally to the consideration of the lower design. Its single GP2015 front-end is unmodified, just like the lower GP2015 front-ends in the other two designs. In this design, however, it is used to receive both the GPS C/A code and the Galileo BOC(1,1) code. This filter is too narrow to pass all of the two main BOC(1,1) lobes, but given the losses of the other two designs, it seemed plausible that the distortion and C/N_0 losses associated with this design might be no worse than those associated with the other designs. In fact, this is exactly what happens. The loss of out-of-band BOC(1,1) power amounts to 2.3 dB, and distortion mismatch causes an additional 2.3 dB of processing loss to yield a total loss of 4.6 dB. Thus, the lower RF front-end of Fig. 2 performs no worse than the other front-ends.

Note that the 2.3 dB distortion loss could be avoided if the receiver could use a replica of the band-limited BOC(1,1) signal in its correlation and accumulation calculations. A practical receiver, however, must correlate the received signal with a replica of the non-band-limited BOC(1,1) signal because it is much easier to generate.

The third front-end has been chosen for the final design. There is no reason to prefer either of the other two designs based on C/N_0 loss considerations. The upper front-end is the least desirable because it involves the use of 2 GP2015 units and double the computational load on the software correlator. The middle front-end has the disadvantage of requiring a second GP2015 and a frequency divider, but it has the simplest signal

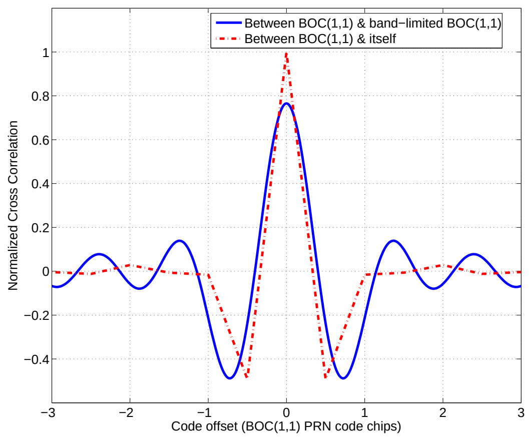


Fig. 4. Distortion of BOC(1,1) correlation function due to band-limiting in the RF front end.

processing in its software correlators. The lower front-end has the simplest RF hardware design: that of an L1 C/A-code receiver. Its software correlation calculations are somewhat more complex than those of the middle design, but this additional complexity has not been deemed significant enough to outweigh its advantage in terms of reduced hardware complexity and the elimination of the need to develop any new hardware.

Figure 4 illustrates the distortion of the BOC(1,1) signal that is caused by the band-limiting in the chosen RF front-end. The red curve is the BOC(1,1) signal's autocorrelation function for the GIOVE-A L1-B signal ii. The blue curve is the cross correlation between the original BOC(1,1) signal and the band-limited version that comes out of an FFT-based model of the RF front-end that has a "brick-wall" filter with a bandwidth of 1.9 MHz. The blue curve is what will be sensed by a receiver when doing acquisition and DLL code phase tracking. It has significant attenuation of its peak, significant displacement of its side-lobes, and ringing, but it retains a distinct enough central peak to be useful for acquisition and for development of a code phase discriminator for a DLL. The shape of this curve can be used to develop a discriminator based on early, late, and prompt accumulations which has an output that approximately equals the code phase error measured in PRN code chips.

V. REVIEW OF BIT-WISE PARALLEL CORRELATION AND ACCUMULATION

The software receiver tested here uses bit-wise parallel Boolean logic computations to calculate correlations. This approach differs from other software receivers that perform integer-based computations in either the time domain or frequency domain. Bit-wise parallel correlation processes 32 RF output data samples at a time. This reduces the number of instructions by a factor of 2 or more [4]. Bit-wise parallel correlation is similar to how a hardware correlator functions. The bit-wise nature of this correlation method is aided by how the RF front end output data is read into the PC's memory. 32-bit words, representing 8 samples from the L1 front end and 8 from the L2 front end, are synchronously read into the PC's memory. There is no translation to an integer representation of the signal. Boolean logic operates on 32-bit words, each of which represents the sign or magnitude of various signals at 32 successive samples.

The software receiver requires both code and carrier replicas. The code replicas are computed in real-time using the method described in the next section. In order to reduce the computational load during correlation, the carrier replicas are pre-computed and stored in memory. In-phase and quadrature L1 and L2 carrier signals are computed on a coarse frequency grid with respective 175 Hz and 87.5 Hz spacings. The initial phase offset for these replicas is 0. These replicas require a total of 1328 kilobytes of storage. The correlator compensates for the use of carrier replica signals of the incorrect frequency and phase. The in-phase and quadrature accumulations are rotated by an angle that represents the average phase difference between the available base-band signal and the intended signal at the correct frequency and phase. This procedure is thoroughly explained in [4], [5], and [6].

Base-band mixing is performed by computing the bit-wise exclusive-or of the RF data sign words and carrier replica sign words over a C/A code period. For the Galileo L1-B BOC(1,1) codes, accumulations are computed over 4092-chip intervals which are nominally 0.004 secs. The RF data magnitude words and carrier replica magnitude words are not operated on, and are simply passed to the next stage. The resultant base-band signal has a 3-bit representation consisting of a 32-bit sign word, a 32-bit low magnitude word, and a 32-bit high magnitude word for 32 successive samples. Code mixing for the C/A code also involves a sequence of bit-wise exclusive-or operations. To mix the base-band signal with the C/A code, the base-band sign words are exclusive-ored with the code replica's sign words. The resultant signal also has a 3-bit representation.

VI. REVIEW OF REAL-TIME GENERATION OF OVER-SAMPLED PRN CODES

This section reviews the method for real-time generation of bit-wise parallel representations of the over-sampled versions of the required PRN codes. The software correlator needs to generate bit-wise parallel representations of the prompt and early-minus-late over-sampled PRN codes, because they are too long to pre-compute and store in memory.

The real-time generation of over-sampled bit-wise parallel PRN codes requires several values: the length of a PRN code chip, the sample interval, the early-minus-late delay, the end time of the first code chip relative to the first sample time, and the actual +1/−1 values of the PRN code chips. This generation method of the bit-wise parallel PRN codes is dependent on the assumption that the sampling frequency, the chipping rate, and the early-minus-late code delay are all constant.

Over-sampling and translation into a bit-wise parallel representation are accomplished simultaneously through the use of three pre-computed tables, one for the prompt code, one for the magnitude of the early-minus-late code, and one for the sign of the early-minus-late code. The three tables' size is relatively small and is independent of the PRN code period. In fact, it can be reused for PRN codes in a multi-channel receiver. The PRN code chip length, the sample interval, the early-minus-late delay, and a desired code phase resolution are used to pre-compute the look-up tables. This tables contains the bit-packed 32-bit representations for all the possible PRN code combinations and initial phase offsets found in 32 successive samples.

The tables have 2 variable inputs: the sequence of code chip values that span a 32-bit data word and the end time of the initial prompt chip relative to the data word's first sample. A calculation is made using these values that gives an index into the 3 tables, which in turn outputs the correct bit-wise parallel representation of the prompt and early-minus-late codes. An exhaustive description of this technique is given in [7]. It was applied to a civilian L1/L2 software reciever in [6].

The bitwise-parallel software correlation calculations require a bitwise-parallel representation of each over-sampled BOC(1,1) code. There are two methods for generating such representations. The first method pre-computes tables of such representations for entire code periods i. The second method computes these representations in real-time by using a tabulated function ii. The first method executes more rapidly, but it requires more storage, and its storage requirement grows linearly with the length of the PRN code. The 4 msec Galileo L1-B PRN code is long enough to warrant using the second method in order to save memory.

5 illustrates a section of a BOC(1,1) code and the corresponding bitwise-parallel representation of the over-sampled versions that are needed by the software correlator. The four digital time histories shown in the figure are, from top to bottom, the prompt, early, late, and

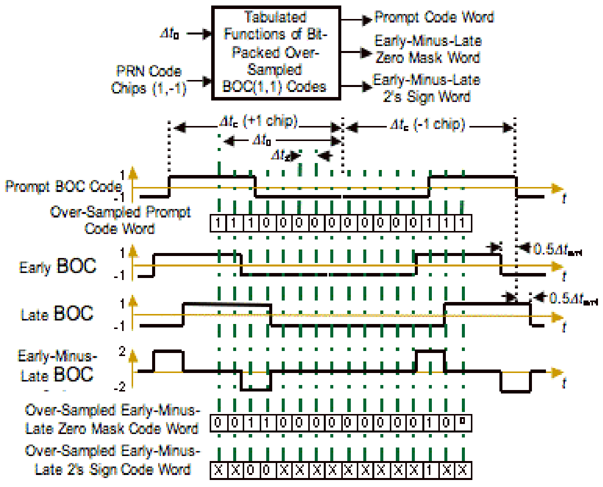

Δt_0
PRN Code Chips (1,-1)
Tabulated Functions of Bit-Packed Over-Sampled BOC(1,1) Codes
Prompt Code Word
Early-Minus-Late Zero Mask Word
Early-Minus-Late 2's Sign Word
Δt_c (+1 chip)
Δt_c (-1 chip)
Δt_0
Δt_s
Prompt BOC Code
Over-Sampled Prompt Code Word
1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1
Early BOC
0.5Δt_{eml}
Late BOC
0.5Δt_{eml}
Early-Minus-Late BOC
Over-Sampled Early-Minus-Late Zero Mask Code Word
0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0
Over-Sampled Early-Minus-Late 2's Sign Code Word
X X 0 0 X X X X X X X X X 1 X X
t

Fig. 5. A bit-packed over-sampled BOC(1,1) code and its real-time generation via tabulated functions.

early-minus late versions of the BOC(1,1) code. Notice how the prompt code includes two full chip periods of length Δt_c with a +1 chip followed by a -1 chip. Each chip is mixed with one +1/-1 oscillation period of the sine-phased BOC. The green vertical dash-dotted lines indicate 16 sample times with sample spacing Δt_s. Representations of the values of the prompt and early-minus-late BOC(1,1) codes at these sample times are packed into three 16-bit data words using bit-wise parallelism. These words are the over-sampled prompt code word, the over-sampled early-minus-late zero mask code word, and the over-sampled 2's sign code word. The block with inputs and outputs at the top of the figure indicates that these three words can be generated as functions of two inputs: the end time of the first BOC(1,1) PRN code chip measured relative to the time of the first sample, Δt_0, and the actual PRN code chip values that span the samples in question (a +1 followed by a -1 for this example).

Ref. [7] explains how to compute and use the real-time code generation table of Fig. 5 when operating on BPSK PRN codes such as the GPS L1 C/A code. It is possible, in theory, to directly apply the methods of [7] to BOC(1,1) codes. In this scenario, the BOC(1,1) signal is treated as a straight PRN code that chips at twice the chipping rate and that has twice as many chips. Thus, the BOC(1,1) signal of Fig. 5 would be modeled as being a PRN code with a chipping rate of 2.046 MHz, which would halve its Δt_c value, and it would be modeled as having 4 chips during the 16-sample interval in question: +1,-1,-1,+1. Unfortunately, the size of the tables that generate the real-time over-sampled codes grows dramatically if one uses this approach. This growth occurs because this approach tabulates $2^4 = 16$ possibilities for the chip values even

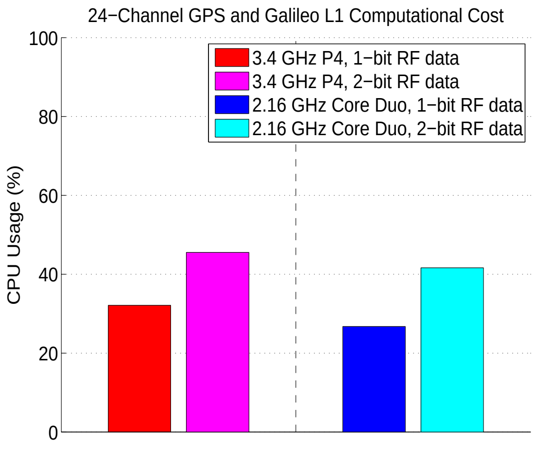


Fig. 6. Computational performance comparison processing either 2-bit or 1-bit RF data streams on either a 3.4 GHz Intel Pentium 4 microprocessor or a dual-core 2.16 GHz Pentium Core Duo microprocessor.

though the BOC(1,1) nature of the signal implies that only $2^2 = 4$ of these possibilities can be realized.

The required table size for real-time generation of the over-sampled BOC(1,1) code has been greatly reduced through the development of special forms of the tabulated functions, forms that account for the BOC(1,1) structure. Appendix A describes the modifications to the algorithms of Re that are needed in order to generate these special tables.

VII. COMPUTATIONAL LOAD AND MEMORY REQUIREMENTS

The computational load and memory requirements of a software receiver are two important factors. The software receiver described here has a significant computational load, demanding a high-end PC or DSP chip in order to track 24 or more channels. Its memory requirements are modest, requiring only a few megabytes of storage for both the code and data.

Computational Load

The bit-wise parallel correlation and accumulation calculations use mostly simple logic and table look-up operations in order to form the 4 accumulations for each PRN code. Generating the over-sampled PRN codes requires additional computations. A description of the computational requirements in terms of instructions per accumulation can be found in [6].

Fig. 6 shows the computational performance requirements for the software receiver to track 12 GPS L1 C/A code signals and 12 Galileo L1-B BOC(1,1) signals in real time. The figure demonstrates the computational requirements of bit-wise parallel correlation using 32-bit signal processing and either 2-bit or 1-bit RF data streams. Two different microproccessors were tested. The first is an Intel Pentium 4 running at 3.4 GHz, and the second is a dual-core Intel Core Duo, with each core running at 2.16 GHz.

The cases that show processing of 2-bit and 1-bit RF data streams demonstrate the reduction in complexity when going from 2 bits to 1 bit when computing the bit-wise parallel accumulations. A reduction of less than a factor of 2 is expected when reducing the RF front end bits from a 2-bit digitization to a 1-bit digitization and using the real-time generation of the over-sampled PRN codes [6]. The computational requirements of the bit-wise parallel processing are dependent on the number of RF data bits, but the generation of over-sampled PRN codes is independent of the number of RF data bits. This translates into a decrease of 25%, as seen in Fig. 6, when going from 2-bit RF front end data streams to 1-bit RF front end data streams. Note that roughly a 2-dB loss in C/N_0 occurs when reducing the number of RF front end data bits from a 2 to 1 [9].

Note that this algorithm can be adapted to work with a different number of bits in the representation of the RF front end output data and of the cosine and sine mixing signals. Increases in the number of bits tends to increase the complexity of the bit-wise parallel correlation, and, in turn, the computational requirements. Using more than 3 bits with the bit-wise parallel algorithms tends to degrade the performance enough, that algorithms using fixed-point multiply-and-accumulate operations will win.

To produce the results from the dual-core microprocessor in Fig. 6, a non-real-time version of the software receiver code was modified. This was performed as a last-minute investigation into the ease of re-writing parts of the code into a multi-threaded application and to demonstrate its performance on newer dual-core microprocessors. The performance results from the dual-core microprocessor illustrate the ease at which performance gains can be achieved by parallelizing GNSS signal processing code. The 2-bit RF front end data case requires about 41% of each 2.16 GHz core to run in real time. Thus, the two independent cores act like a single core running at roughly twice the speed of each indi-vidual core, giving the 11% reduction in computational cost. Note, that if the scaling was perfect, a 27% decrease in computational cost would have been realized. Oher factors such as memory speed may explain this discrepancy. In summary, these results illustrate how parallelizable GNNS correlator signal processing is.

Memory Requirements

The pre-computed base-band mixing signals and PRN code over-sampling tables require a certain amount of memory. Each replica base-band mixing signal must occupy 180 32-bit words for the L1 C/A code signal acnd 715 32-bit words for the Galileo L1-B code signal. These sizes guarantee covering the full 5,714 and 22,857 RF front end samples for, respectively, the 1 ms L1 C/A code accumulation and 4 ms Galileo L1-B code accumulation intervals. Thus, 180×4=720 bytes are required for each bit of each carrier signal that must be stored for the C/A code signal, and 715×4=2860 bytes are required for each bit of each carrier signal that must be stored for the CM/CL code signal. The sine and cosine signals each have two-bit representations, which translates into a total storage requirement of 2880 bytes for the GPS L1 carrier replicas and 11,440 bytes for the Galileo L1 carrier replicas. For the L1 C/A code signal, there are 115 Doppler shifts that must be stored in order to cover the –10 KHz to +10 KHz range with a 175 Hz grid spacing. For the Galileo L1-B signal, 230 Doppler shifts must be stored to cover the –10 KHz to +10 KHz range with a 87.5 Hz grid spacing. Since the Galileo L1-B signal has a longer period, it is advisable to use a finer grid size for the carrier replicas. This translates into 2893 kilobytes of storage for all of the carrier replica signals, which is a substantial amount.

One significant improvement in storing the carrier replicas is possible. The GPS and Galileo L1 carrier replicas are spaced about the same intermediate frequency, but have different lengths and different grid sizes. The proposed method to reduce the size of the table is to create a set of carrier replicas with 1-ms lengths with Doppler shifts covering the range –10 KHz to +10 KHz using a 87.6-Hz grid spacing. These are adequate for the shorter GPS L1 C/A code accumulation interval. However, for the Galileo L1-B accumulation interval of 4 ms, one has to perform 4 non-coherent accumulations over 1-ms intervals, and compute a fix-up to the in-phase and quadrature accumulations before summing them together to produce an effective 4 ms coherent accumulation. The fix-up is a simple rotation on the 1-ms accumulations which is a function of the desired initial phase offset and the actual initial phase offset of the carrier replica. This new set of carrier would have a size of 646 kilobytes, which is a reduction in memory size by a factor of 5 over the above approach. The only downside is a very small increase in the computational cost due to the fix-up of the accumulations.

The GPS L1 C/A and Galileo L1-B BOC(1,1) code look-up tables require a modest amount of memory. The required memory is given by equation (56) in [6]. The equation is not repeated here because it requires explanation of a number of variables that are beyond the scope of this review. The required memory scales linearly with the code phase resolution and exponentially with the number of samples per data word.

Three PRN code tables are needed for the GPS L1 C/A code, and three are needed for the Galileo L1-B BOC(1,1) code. The need for separate tables is for the GPS and Galileo signals is because the GPS L1 C/A code chips at 1.023×10^6 chips/sec, and the Galileo L1-B signal chips at 2.046×10^6 chips/sec. For the GPS L1 C/A code, the early-minus-late code spacing is one-half a chip, and there are 32 samples per data word. This software receiver uses the following parameters: an RF sampling interval of 175 nsec and the range-equivalent code phase resolution is 7.50 m. These values yield at most 8 code chips per data word, 42 tabulated code start/stop times, and 3584 entries per PRN code table. These three PRN code tables together require 42 kilobytes of memory. For the Galileo L1-B BOC(1,1) code, the early-minus-late code spacing is one-third a chip, and there are 32 samples per data word. The RF sampling interval and code phase resolution are the same as above. These values yield at most 7 code chips per data word, 42 tabulated code start/stop times, and 1792 entries per PRN code table. These three PRN code tables together require 21 kilobytes of memory.

One method of simplifying the real-time generation of sampled codes is to make use of the fact that the C/A code and the L1-B BOC(1,1) code are relatively short. It is possible to use pre-computed tables of the C/A code chips and L1-B BOC(1,1) code chips with this method. The advantage is that the computational requirements are slightly decreased in favor of a modest increase in the required memory. The total additional memory required to store the auxiliary tables for all 32 GPS satellites and 32 Galileo satellites is 288 kilobytes. To conserve memory, it may be useful in the future to use Galileo PRN code generators in real time. A PRN code generator was not available for GIOVE-A L1-B signal at the time of writing.

A circular buffer stores the most recent 21 msec of RF front end data from each RF front end data. This buffer occupies 29.2 kilobytes of memory.

The total amount of memory required for storing the tables is 3337 kilobytes. The machine code and ancillary data for the software receiver requires additional storage on the order of roughly 1500 kilobytes.

VIII. RECEIVER PERFORMANCE RESULTS

Acquisition and tracking performance has been tested using live data from a roof-mounted antenna in Austin, TX, USA. Since only one Galileo satellite, GIOVE-A,

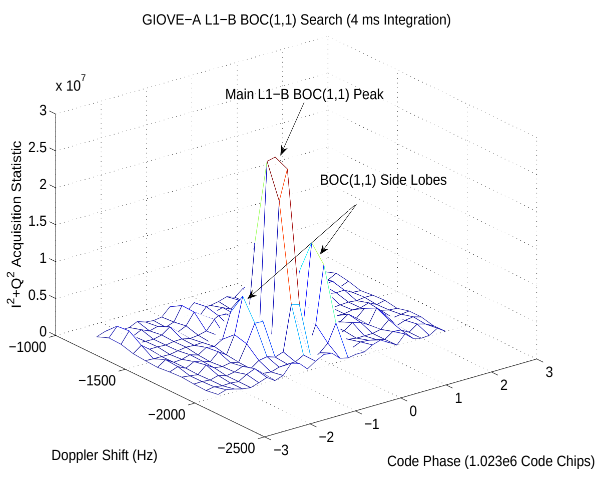


Fig. 7. $I^2 + Q^2$ acquisition statistic versus Doppler shift and code offset for the GIOVE-A L1-B BOC(1,1) signal.

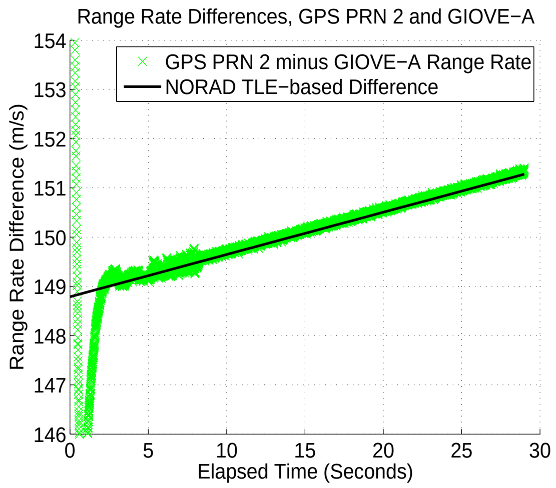


Fig. 9. True and estimated range rate differences between GPS PRN 2 and GIOVE-A.

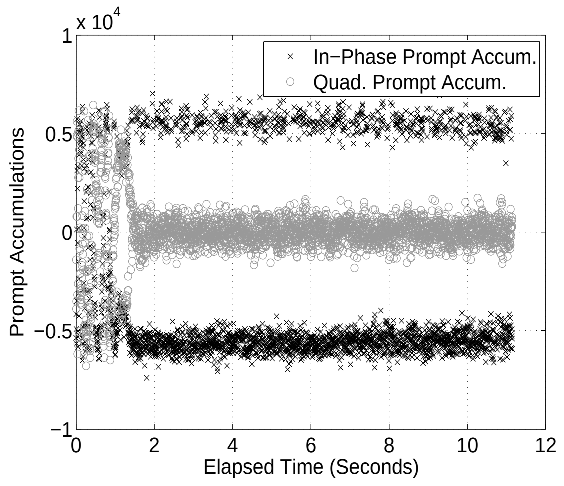


Fig. 8. PLL-baseed tracking of the GIOVE-A signal.

is currently in orbit, and its navigation data bit stream is unknown, navigation performance testing using this satellite was not possible.

Figure 7 shows the prompt $I^2 + Q^2$ acquisition statistic versus Doppler shift and code offset for the GIOVE-A L1-B BOC(1,1) signal. This result is from live data collected in Austin, TX on September 17, 2006. This figure shows the main and side lobes of the BOC(1,1) signal. Note that due to the narrow filter in the RF front end, the BOC(1,1) side lobes have mean peak powers of about 40% of the main lobe and are offset by 0.7 BOC(1,1) chips from the center of the main lobe. If the RF front end had a wider filter, one would expect the side lobes to have mean peak powers of only 25% of the main lobe and have spacings of 0.5 BOC(1,1) chips.

Figure 8 shows the in-phase and quadrature prompt accumulation time history for GIOVE-A over a 11-second interval. The first ∼2 seconds illustrate the transition from acquisition to tracking first with a FLL and then with a PLL. The PLL forces the power in the 250-Hz prompt accumulations into the in-phase component. The transitions from a positive prompt in-phase accumulation to a negative in-phase accumulation, and vice versa, illustrates the data bit transitions in the navigation message. It's clear from this figure that a bias in the distribution of the values of the data bits exists. The distribution of ±1 values is about 25%-75%.

One verification of the tracking performance of a GPS and Galileo software receiver is to investigate common-mode errors that are present on both GPS and Galileo tracking channels. Estimates of range rate of GPS satellites and Galileo satellites will both be biased by the receiver clock drift rate. To investigate this, the difference of the estimated range rate time histories from GPS PRN 2 and GIOVE-A is plotted as the green exes in Figure 9. The true values for the range rates histories can be computed using the broadcast GPS ephemerides and the North American Aerospace Defense Command (NORAD) two-line orbital elements (TLEs) for the GPS satellites and GIOVE-A, respectively. The difference in these true range rate time histories is plotted as the straight black line in Fig. 9. The transient response during the first 3 seconds of the difference in the estimated range rate time histories is caused by pull-in of the PLLs for GPS PRN 2 and GIOVE-A. After this time, it is clear that the two plots coincide well, indicating that a common-mode range rate bias is seen on GPS PRN 2 and GIOVE-A.

IX. SUMMARY AND CONCLUDING REMARKS

A 24-channel real-time GPS and Galileo L1 software receiver that runs on a PC has been implemented and tested. The hardware consists of a COTS GPS L1 C/A code analog-mixing RF front end, a data buffering and acquisition system, and a PC with a 3.4 GHz Intel Pentium 4 processor running RTAI, a real-time variant of Linux. The software consists of GPS and Galileo L1 real-time software correlators and additional software that provides the typical functions such as signal acquisition, signal tracking, and navigation. The software correlator, running on the PC's microprocessor, requires 46% of the CPU's computational capacity. The software receiver was tested with live data from the GPS satellites and GIOVE-A to verify successful acquisition and tracking.

APPENDIX A. ALGORITHMS THAT GENERATE OVER-SAMPLING TABLES FOR BOC(1,1) CODES

This paper uses a modified form of the methods of [7] in order to generate bit-wise parallel representations of over-sampled BOC(1,1) codes in real time. The modifications are needed in order to transition from the BPSK PRN code format assumed in [7] to the BOC(1,1) code format used by the Galileo L1 signal. The only part of [7] that needs modification is the last part of its Section IV. The following modifications are all that are needed: Equations (14)-(16) of [7] are replaced by the following calculations (this description of the modifications uses the same notation as is used in [7] and presumes familiarity with that reference and its definitions of mathematical quantities):

Generate the sine-phased binary-offset carrier for the prompt, early, and late signals:

$$\begin{aligned} B_p(n,i) = \;& mod\left(floor\left\{ 2\left[n-1-\frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right]\right\},2\right) \\ & for\ n = 1,2,3,...,n_s \end{aligned} \tag{3}$$

$$\begin{aligned} B_e(n,i) = \;& mod\left(floor\left\{ 2\left[n-1-\frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right]\right.\right. \\ & \left.\left. +2\left[\frac{\Delta t_{eml}}{2\Delta t_c}\right]\right\},2\right) \quad for\ n = 1,2,3,...,n_s \end{aligned} \tag{4}$$

$$\begin{aligned} B_l(n,i) = \;& mod\left(floor\left\{ 2\left[n-1-\frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right]\right.\right. \\ & \left.\left. -2\left[\frac{\Delta t_{eml}}{2\Delta t_c}\right]\right\},2\right) \quad for\ n = 1,2,3,...,n_s \end{aligned} \tag{5}$$

The 3 binary signals $B_p(n,i)$, $B_e(n,i)$, and $B_l(n,i)$ take on 0 values when the binary-offset carrier equals +1, and they take on 1 values when it equals -1.

Mix the prompt, early, and late PRN code chip values $C_p(n,i)$, $C_e(n,i)$, and $C_l(n,i)$ with the prompt, early, and late binary-offset carrier in order to get the prompt, early, and late sine-phased BOC(1,1) signals:

$$\begin{aligned} BOC_p(n,i) = \;& mod\,[B_p(n,i) + C_p(n,i), 2] \\ & for\ n = 1,2,3,...,n_s \end{aligned} \tag{6}$$

$$\begin{aligned} BOC_e(n,i) = \;& mod\,[B_e(n,i) + C_e(n,i), 2] \\ & for\ n = 1,2,3,...,n_s \end{aligned} \tag{7}$$

$$\begin{aligned} BOC_l(n,i) = \;& mod\,[B_l(n,i) + C_l(n,i), 2] \\ & for\ n = 1,2,3,...,n_s \end{aligned} \tag{8}$$

The 3 binary signals $BOC_p(n,i)$, $BOC_e(n,i)$, and $BOC_l(n,i)$ take on 0 values when the product of the PRN code and the binary-offset carrier equals -1, and they take on 1 values when it equals +1. These values are used to generate the unsigned integer entries of the 3 tables:

$$x_p(i) = \sum_{n=1}^{n_s} BOC_p(n,i) \times 2^{n_s-n} \tag{9}$$

$$\begin{aligned} x_{emlzm}(i) = \;& \sum_{n=1}^{n_s} mod\{[BOC_e(n,i) + BOC_l(n,i)], 2\} \\ & \times 2^{n_s-n} \end{aligned} \tag{10}$$

$$\begin{aligned} x_{eml2s}(i) = \;& \sum_{n=1}^{n_s} mod\{[BOC_e(n,i) + BOC_l(n,i)], 2\} \\ & \times BOC_e(n,i) \times 2^{n_s-n} \end{aligned} \tag{11}$$

where $x_p(i)$, $x_{emlzm}(i)$, and $x_{eml2s}(i)$ are, respectively, the entries of the prompt sign table, the early-minus-late zero-mask table, and the early-minus-late 2's sign table.

ACKNOWLEDGMENTS

This research was supported in part by NASA Office of Space Science through grant No. NNX06AC34G. Madhulika Guhathakurta is the grant monitor.

REFERENCES

[1] M. L. Psiaki, T. E Humphreys, S. Mohiuddin, S. P. Powell, P. M. Kintner, and A. P. Cerruti, “Determination of Galileo GIOVE-A L1 BOC(1,1) PRN Codes,” available on-line at http://gps.ece.cornell.edu/galileo/, April 2006.

[2] D. M. Akos, P.-L. Normark, P. Enge, A. Hansson, and A. Rosenlind, “Real-Time GPS Software Radio Receiver,” in *Proc. of the Institute of Navigation National Technical Meeting*, Long Beach, CA, January 22–24, 2001, pp. 809–816.

[3] J. Thor, P. Normark, and C. Ståhlberg, “A High-Performance Real-Time GNSS Software Receiver and its Role in Evaluating Various Commercial Front End ASICs,” in *Proc. of the Institute of Navigation GPS*, Portland, OR, September 24–27, 2002, pp. 2554–2560.

[4] B. M. Ledvina, M. L. Psiaki, S. P. Powell, and P. M. Kintner, “A 12-Channel Real-Time GPS L1 Software Receiver,” in *Proc. of the Institute of Navigation National Technical Meeting*, Anaheim, CA, January 22–24, 2003, pp. 767–782.

[5] 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) September, 2004.

[6] B. M. Ledvina, M. L. Psiaki, D. J. Sheinfeld, A. P. Cerruti, S. P. Powell, and P. M. Kintner, “A Real-Time GPS Civilian L1/L2 Software Receiver,” in *Proc. of the Institute of Navigation GNSS 2004*, Long Beach, CA, September 21–24, 2004, pp. 986–1005.

[7] M. L. Psiaki, “Real-Time Generation of Bit-Wise Parallel Representations of Over-Sampled PRN Codes,” *IEEE Transactions on Wireless Communications*, Vol. 5, No. 3, March 2006, pp. 487–491.

[8] B. M. Ledvina, F. Mota, and P. M. Kintner, “A Coming of Age for GPS: a RTLinux GPS Receiver,” Workshop on Real Time Operating Systems and Applications, in *Proc. of the Workshop on Real Time Operating Systems and Applications and Second Real Time Linux Workshop* (in conjunction with IEEE RTSS 2000), Orlando, FL, Nov. 27–28, 2000.

[9] A. J. Van Dierendonck, “GPS Receivers,” in *Global Positioning System: Theory and Applications, Vol. I*, B. W. Parkinson and J. J. Spilker Jr. , Eds., American Institute of Aeronautics and Astronautics, (Washington, 1996), pp. 329–407.

[10] Anon., ”GP2015 Miniature GPS Receiver RF Front-End,” Zarlink Semiconductor, Ottawa, Canada, http://products.zarlink.com/product_profiles/GP2015.htm, 2005.