A Real-Time GPS Civilian L1/L2 Software Receiver

B. M. Ledvina, M. L. Psiaki, D. J. Sheinfeld, A. P. Cerruti, S. P. Powell, and P. M. Kintner,
Schools of Electrical and Computer Engineering and Mechanical and Aerospace Engineering, Cornell University

BIOGRAPHY

Brent M. Ledvina is a Postdoctoral Associate in the School of Electrical and Computer Engineering at Cornell University. 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 an Associate 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 GPS technology and applications.

Daniel Sheinfeld is a graduate student in the Department of Aeronautics and Astronautics at Stanford University. He received a B.S. in Mechanical Engineering from Cornell University. His research interest is in the area of automatic controls with applications to aerospace systems.

Alessandro Cerruti is a Ph.D. student in the School of Electrical and Computer Engineering at Cornell University. He received B.S. and M.Eng. degrees in Electrical and Computer Engineering from Cornell University. His current interests include development of GNSS receivers and their applications to studying ionospheric scintillations.

Steven Powell is a Senior Engineer with the Space Plasma Physics Group in the School of Electrical and Computer Engineering at Cornell University. He has been involved with the design, fabrication, testing, and launch activities of many scientific experiments that have flown on high altitude balloons, sounding rockets, and small satellites. He has M.S. and B.S. degrees in Electrical Engineering from Cornell University.

Dr. Paul M. Kintner, Jr. is a professor of Electrical and Computer Engineering at Cornell University. His interests as a rocket scientist range from exploring the northern lights to understanding space weather. His recent work with GPS receiver design and scintillations led NASA to appoint him chair of the Living With a Star Geospace Mission Definition Team.

ABSTRACT

A dual-frequency civilian GPS receiver has been developed that runs 10 tracking channels in real time using a software correlator. This work is part of an effort to develop a flexible receiver that can use the new GPS L2C CM/CL signals as they become available on L2 without the need for new correlator hardware. The receiver consists of an RF front end, a system of shift registers, a digital data acquisition (DAQ) system, and software that runs on a 3.2 GHz PC. The direct RF sampling front end uses intentional aliasing and converts the GPS L1 C/A codes and L2 CM and CL codes into a 1-bit digital data stream sampled at a frequency between 8 and 12.5 MHz. The shift registers parallelize the 1-bit data stream, which the DAQ reads into the PC's memory using direct memory access. The PC performs base-band mixing and PRN code correlations in a manner that directly simulates a hardware digital correlator. It also performs the usual signal tracking and navigation functions, under the control of a real-time Linux operating system.

The main contribution of this work is a method for real-time generation of over-sampled bitwise-parallel representations of PRN codes via a table look-up function. The over-sampled PRN codes need to be in a bitwise-parallel format in order to be used in the software receiver's bitwise-parallel correlation algorithms. On-line generation of the over-sampled codes is required because it is impractical to pre-compute and store over-sampled representations of very long codes, such as the L2 CL code.

The GPS civilian L1/L2 software receiver tracks 10 channels in real time and has a navigation accuracy of 2–5 meters. It requires 80% of the processing capabilities of a 3.2 GHz Intel Pentium 4 PC.

I. INTRODUCTION

A real-time global navigation software receiver provides distinct advantages in an evolving signal and code environment. The current Global Positioning System (GPS) is slated to expand its capabilities to include

new civilian codes on the L2 frequency and a new L5 frequency. A receiver that uses a hardware digital correlator will require hardware modifications in order to use these new signals. In the near term, a receiver designer will be faced with a complex trade-off in order to decide whether the extra complexity is worth the improved performance that will accrue only very slowly as new GPS satellites replace older models. A software receiver can use 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. Furthermore, a software receiver can be reprogrammed to use the Galileo system, GLONASS, or both, which provides an added benefit from the use of a software radio architecture. Thus, there are good reasons to develop practical real-time software GPS receivers.

This work focuses on the development of a dual-frequency GPS software receiver that utilizes the L1 C/A code and the new L2 civilian CM/CL code. This receiver will be useful for performing dual-frequency ionospheric corrections. It will also be useful for studying the ionosphere.

A hardware dual-frequency GPS receiver can be broken down into various components. First, a dual-frequency antenna, possibly followed by a pre-amp, receives the two L-band GPS signals. After the antenna comes an RF section that filters and down converts the GHz GPS 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 10 or more channels. For each satellite and each carrier frequency, the correlator mixes the Doppler-shifted intermediate frequency signal to base-band 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 message, and that compute the navigation solution.

A software receiver differs from a standard hardware receiver in one distinct way. The functions of the correlator chip are moved to software that runs on a general-purpose microprocessor. This move changes the layout of the receiver, see Fig. 1. 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 dual-frequency correlator shown in Fig. 1 processes inputs from one or two data streams containing the signals at the two GPS frequencies.

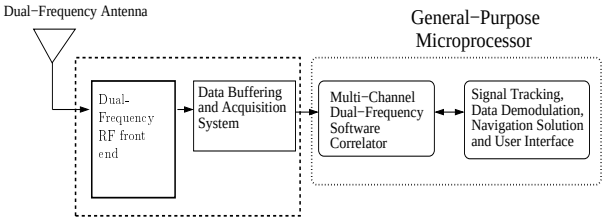

Fig. 1. A typical GPS software receiver showing the separation between special purpose hardware and general hardware.

GPS software receivers have received a lot of interest recently [1], [2], [3], [4]. The work presented in this paper improves upon the software receiver of [3] and [4] in two ways. First, the software receiver discussed here is dual-frequency, thus enabling a more capable receiver able to produce a more accurate navigation solution. Second, this paper explains how to efficiently generate over-sampled PRN codes in real time. This capability is required in order to adapt the software correlator method of [3] and [4] to use long PRN codes, such as the CL code, without using unreasonable amounts of memory.

The remaining portions of this paper explain the internal workings of a 10-channel real-time GPS civilian L1/L2 software receiver and present experimental performance results for this system. The second section describes the system configuration. The third section reviews the structure of spread-spectrum signals and standard correlation accumulation calculations. Section IV reviews the bit-wise parallel software correlation technique of [3] and [4]. Section V describes real-time generation of bit-wise parallel over-sampled PRN codes. Section VI discusses computational and memory requirements of the software correlator. Section VII presents receiver performance results. Section VIII discusses future work. The last section gives a summary and concluding remarks.

II. SYSTEM CONFIGURATION

A principal component of the GPS software receiver is a personal computer. The current system uses a PC with a 3.2 GHz Intel Pentium 4 processor running the Real Time Application Interface (RTAI) 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 very 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 easy to develop, test, debug, and run real-time software.

The next component of the software receiver is the RF front end. It is a direct RF sampling front end similar to the one described in [5]. The heart of this front end is an analog-to-digital converter (ADC) that can function with an input bandwidth of up to 2.2 GHz and that can perform 8-bit conversion at continuous sample rates up to 1 GHz. The ADC samples the GPS signals at 12.199 MHz, which aliases the L1 carrier down to a nominal frequency of 1.749 MHz and the L2 carrier down to a nominal frequency of –4.499 MHz. Each 8-bit sample gets processed by a separate logic unit to create a sign bit after subtraction of an appropriate bias that minimizes the signal-to-noise ratio's digitization loss.

A data acquisition system reads the digitized sign 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 sign bits into a 32-bit word. A divide-by-32 counter converts the 12.199 MHz clock down to 381.22 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 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.

The software receiver is written entirely in ANSI C using tools available from standard Linux distributions. To promote portability of the software, no processor-specific assembly language or special instructions are used.

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 [6], 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.

Three new developments to the Cascade software were required for it to operate with a dual-frequency correlator. First, a set of code and carrier tracking loops for the L2 CM signal were developed. The Cascade software already has a delay-locked loop for code tracking and a frequency-locked loop for carrier tracking. These loops are for tracking the L1 C/A code signal. Modified versions of these loops were implemented for the L2 CM signal. The modifications involved changing the nominal carrier frequency from L1 to L2 and adjusting the integration interval to match the nominal 0.002 second accumulation interval of the CM code. Code and carrier tracking loops for the L2 CL code have not yet been developed.

Second, a method has been developed to remove the ionospheric delay from the L1 C/A code pseudorange measurement. The method involves low-pass filtering the high bandwidth difference of the L2 CM code and L1 C/A code start times to produce an estimate of the differential code delay. This estimate is output at 0.1 Hz. In the Cascade software, the estimated differential code delay is re-scaled to an equivalent range delay at the L1 carrier frequency. The L1 C/A code pseudorange is then corrected by this equivalent range delay. Alternate methods for performing this correction can be found in [7].

Third, an acquisition method for detecting the L2 CM/CL signal has been developed. The Cascade software uses a brute-force code phase offset and Doppler frequency search routine for C/A code acquisition. Since the CL and CM code periods are respectively 20 and 750 times longer than the C/A code period, it is easier to acquire the C/A code first if the signal is strong. The modified Cascade software first acquires the L1 C/A code, and the L1 carrier Doppler shift is used to program the L2 carrier NCO. The correct L2 carrier frequency is determined via re-scaling of the acquired L1 Doppler shift by the ratio of the L2 to the L1 frequencies. This reduces the CM acquisition search space to one dimension, the code start time dimension. The CM acquisition then carriers out a brute-force search for the start time. In the future a better acquisition strategy will be implemented, one that recognizes that the L2 CM code start time can fall only within a limited range of offsets from the L1 C/A code start time.

One missing component of the Cascade software is a data demodulation routine for the convolutionally-coded 50 symbol per second data stream on the CM code. This data stream provides ephemeris and clock information that is also available on the L1 C/A code 50 bit per second data stream. Because the receiver is configured to track the C/A code, it is not necessary to decode the CM code data stream.

The dual-frequency software correlator has been designed as an independent software module that interacts with other parts of the receiver according to well-defined interface specifications. This modular approach provides flexibility in the internal workings of the receiver. One benefit of this modularity is that the mixing methods and correlation routines are transparent to the other standard software modules. This enables quick changes in correlator design that do not significantly affect other parts of the GPS receiver. Another benefit is that the receiver can be reconfigured on the fly to operate as a single-frequency L1 C/A code receiver.

III. REVIEW OF THE SPREAD SPECTRUM GPS SIGNAL AND CORRELATION ACCUMULATIONS

The direct RF sampling front end outputs a single RF data stream that contains both the L1 and L2 signals. The following is a model of the output data stream of the RF front end:

$$
\begin{aligned}
y(t_i) = \sum_j \Bigg(& A_{L1j} D_{L1jk} C_{L1j} \\
& \left[0.001 \left(\frac{t_i - \tau_{L1jk}}{\tau_{L1jk+1} - \tau_{L1jk}} \right) \right] \\
& \times \cos[\omega_{IF,L1} t_i + \phi_{L1j}(t_i)] \\
& + A_{L2j} \Bigg\{ D_{L2jk} C_{L2M0jk} \left[0.001 \right. \\
& \left. \times \left(\frac{t_i - \tau_{L2jk}}{\tau_{L2jk+1} - \tau_{L2jk}} \right) \right] \\
& + C_{L20Ljk} \left[0.001 \left(\frac{t_i - \tau_{L2jk}}{\tau_{L2jk+1} - \tau_{L2jk}} \right) \right] \Bigg\} \\
& \times \cos[\omega_{IF,L2} t_i - \phi_{L2j}(t_i)] \Bigg) + n_j
\end{aligned}
\tag{1}
$$

where t_i is the sample time, the subscript j refers to a particular GPS satellite, A_{L1j} is the L1 signal amplitude for satellite j, A_{L2j} is the L2 signal amplitude, D_{L1jk} is the L1 signal navigation data bit, D_{L2jk} is the L2 signal navigation data bit, $C_{L1j}[t]$ is the C/A code, $C_{L2M0jk}[t]$ is the CM code applicable in the nominal 1 ms time span from τ_{L2jk} to τ_{L2jk+1} interspersed with 0s every other chip, C_{L20Ljk} is the CL code applicable on this same interval interspersed with 0s every other chip on the opposite chips to the zeros from the CM code, τ_{L1jk} and τ_{L1jk+1} are the start times of the received k^{th} and $k{+}1^{st}$ C/A code periods, τ_{L2jk} and τ_{L2jk+1} are the start and stop times of the received k^{th} 1023 chip interval of the CM/CL code, $\omega_{IF,L1}$ is the intermediate frequency corresponding to the L1 carrier frequency, $\omega_{IF,L2}$ is the intermediate frequency corresponding to the L2 carrier frequency, $\phi_{L1j}(t_i)$ is the L1 carrier phase perturbation due to accumulated delta range and ionospheric effects, $\phi_{L2j}(t_i)$ is the L2 carrier phase perturbation due to accumulated delta range and ionospheric effects, and n_j is the receiver noise. The summation is over all visible GPS satellites. The negative sign in front of $\phi_{L2}(t_i)$ comes from the effective high-side L2 mixing that occurs in the RF front end that has been used. The C/A code has a length of 1023 chips and a chipping rate of 1.023 MHz. The CM code has a length of 10230 chips and the CL code has a length of 767,250 chips. These latter two codes each have a chipping rate of 511.5 KHz. The combined CM/CL code contains time-division multiplexed chips from the CL and CM codes, and has an equivalent chipping rate of 1.023 MHz [8]. Fig. 2 illustrates an example of the multiplexed CM/CL code and the C/A code.

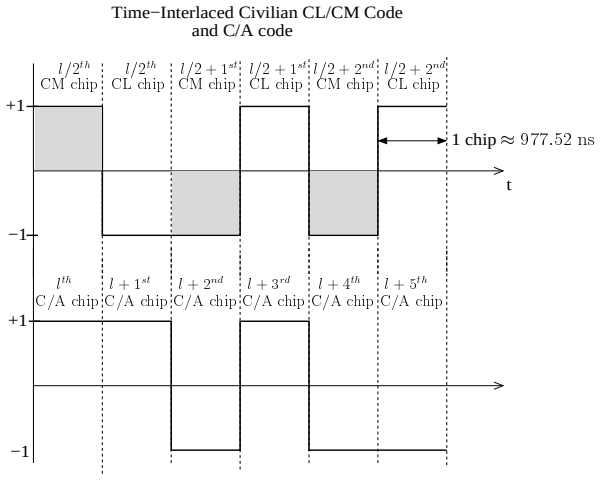

Fig. 2. Illustration of the time-division multiplexed L2 CM/CL code and the L1 C/A code.

A second type of RF data stream is possible. Instead of a single data stream, two RF front ends could output two separate data streams, one containing the L1 signal and the other containing the L2 signal. The two RF data streams would be processed independently. Using an RF data stream of this type would separate eq. (1) into two components, one containing the L1 C/A code signal and the other containing the L2 CM/CL code signal. Although the remainder of this paper assumes a single dual-frequency data stream in the form of eq. (1), the methods used can be applied equally well when there are two independent data streams.

A GPS receiver works with correlations between the received signal and a replica of it. The correlations are used to acquire and track the signal. A replica is required for each satellite. The replica is composed of two parts, the carrier replica and the PRN code replica.

Two carrier replica signals are used, an in-phase signal and a quadrature signal. When mixed with the code replica they form the in-phase and quadrature replicas. The in-phase and quadrature replicas for the L1 C/A code signal are

$$\begin{aligned} y_{I,L1j}(t_i) = \; & C_{L1j}\left[0.001\left(\frac{t_i - \hat{\tau}_{L1jk}}{\hat{\tau}_{L1jk+1} - \hat{\tau}_{L1jk}}\right)\right] \\ & \times \cos\{\omega_{IF,L1} t_i + [\hat{\phi}_{L1jk} \\ & + \hat{\omega}_{Dopp,L1jk}(t_i - \hat{\tau}_{L1jk})]\} \end{aligned} \tag{2}$$

$$\begin{aligned} y_{Q,L1j}(t_i) = \; & C_{L1j}\left[0.001\left(\frac{t_i - \hat{\tau}_{L1jk}}{\hat{\tau}_{L1jk+1} - \hat{\tau}_{L1jk}}\right)\right] \\ & \times \sin\{\omega_{IF,L1} t_i + [\hat{\phi}_{L1jk} \\ & + \hat{\omega}_{Dopp,L1jk}(t_i - \hat{\tau}_{L1jk})]\} \end{aligned} \tag{3}$$

where equations (2) and (3) apply during the k^{th} C/A code period. In these equations $\hat{\tau}_{L1jk}$ and $\hat{\tau}_{L1jk+1}$ are the receiver's estimates of the start times of the k^{th} and $k+1^{st}$ code periods, $\hat{\phi}_{L1jk}$ is the estimated L1 carrier phase at time $\hat{\tau}_{L1jk}$, and $\hat{\omega}_{Dopp,L1jk}$ is the estimated L1 carrier Doppler shift during the k^{th} code period.

The in-phase and quadrature replicas for the L2 CM code signal interspersed with zeros are

$$\begin{aligned} y_{I,L2j}(t_i) = \; & C_{L2M0jk}\left[0.001\left(\frac{t_i - \hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+1} - \hat{\tau}_{L2jk}}\right)\right] \\ & \times \cos\{\omega_{IF,L2} t_i - [\hat{\phi}_{L2jk} \\ & + \hat{\omega}_{Dopp,L2jk}(t_i - \hat{\tau}_{L2jk})]\} \end{aligned} \tag{4}$$

$$\begin{aligned} y_{Q,L2j}(t_i) = \; & C_{L2M0jk}\left[0.001\left(\frac{t_i - \hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+1} - \hat{\tau}_{L2jk}}\right)\right] \\ & \times \sin\{\omega_{IF,L2} t_i - [\hat{\phi}_{L2jk} \\ & + \hat{\omega}_{Dopp,L2jk}(t_i - \hat{\tau}_{L2jk})]\} \end{aligned} \tag{5}$$

where eqs. (4) and (5) apply during the k^{th} CM/CL code 1023-chip interval. In these equations $\hat{\tau}_{L2jk}$ and $\hat{\tau}_{L2jk+1}$ are the receiver's estimates of the start times of the k^{th} and $k+1^{st}$ code intervals, $\hat{\phi}_{L2jk}$ is the estimated L2 carrier phase at time $\hat{\tau}_{L2jk}$, and $\hat{\omega}_{Dopp,L2jk}$ is the estimated L2 carrier Doppler shift during the k^{th} 1023-chip interval. The replicas for the L2 CL code signal interspersed with zeros can be formed by replacing $C_{L2M0jk}[t]$ in equations (4) and (5) with $C_{L20Ljk}[t]$.

A typical dual-frequency receiver computes the estimates $\hat{\tau}_{L1jk}$, $\hat{\tau}_{L1jk+1}$, $\hat{\phi}_{L1jk}$, $\hat{\omega}_{Dopp,L1jk}$, $\hat{\tau}_{L2jk}$, $\hat{\tau}_{L2jk+1}$, $\hat{\phi}_{L2jk}$, and $\hat{\omega}_{Dopp,L2jk}$ by various means described in [9]. These include open-loop acquisition methods and closed-loop signal tracking methods such as a delay-locked loop to compute $\hat{\tau}_{L1jk}$ and $\hat{\tau}_{L1jk+1}$ and a phase-locked loop or a frequency-locked loop to compute $\hat{\phi}_{L1jk}$ and $\hat{\omega}_{Dopp,L1jk}$. The software receiver developed here uses standard techniques for forming these estimates. These techniques are not discussed in detail here.

The receiver uses the carrier and code replicas to compute the following L1 C/A code in-phase and quadrature correlation accumulations:

$$\begin{aligned} I_{L1jk}(\Delta) = \; & \sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L1j}\left[0.001\right. \\ & \left.\times\left(\frac{t_i + \Delta - \hat{\tau}_{L1jk}}{\hat{\tau}_{L1jk+1} - \hat{\tau}_{L1jk}}\right)\right] \\ & \times \cos\{\omega_{IF,L1} t_i + [\hat{\phi}_{L1jk} \\ & + \hat{\omega}_{Dopp,L1jk}(t_i - \hat{\tau}_{L1jk})]\} \end{aligned} \tag{6}$$

$$\begin{aligned} Q_{L1jk}(\Delta) = \; & \sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L1j}\left[0.001\right. \\ & \left.\times\left(\frac{t_i + \Delta - \hat{\tau}_{L1jk}}{\hat{\tau}_{L1jk+1} - \hat{\tau}_{L1jk}}\right)\right] \\ & \times \sin\{\omega_{IF,L1} t_i + [\hat{\phi}_{L1jk} \\ & + \hat{\omega}_{Dopp,L1jk}(t_i - \hat{\tau}_{L1jk})]\} \end{aligned} \tag{7}$$

where i_k is the index of the first RF front end sample time that obeys $\hat{\tau}_{L1jk} \le t_{i_k}$ and $N_k + 1$ is the total number of samples that obey $\hat{\tau}_{L1jk} \le t_i < \hat{\tau}_{L1jk+1}$. The time offset Δ causes the replica PRN code to play back early if it is positive and late if Δ is negative.

The L2 CM code in-phase and quadrature correlation accumulations are

$$\begin{aligned} I_{L2CMjk}(\Delta) = \; & \sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L2M0jk}\left[0.002\right. \\ & \left.\times\left(\frac{t_i + \Delta - \hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+2} - \hat{\tau}_{L2jk}}\right)\right] \\ & \times \cos\{\omega_{IF,L2} t_i - [\hat{\phi}_{L2jk} \\ & + \hat{\omega}_{Dopp,L2jk}(t_i - \hat{\tau}_{L2jk})]\} \end{aligned} \tag{8}$$

$$\begin{aligned} Q_{L2CMjk}(\Delta) = \; & -\sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L2M0jk}\left[0.002\right. \\ & \left.\times\left(\frac{t_i + \Delta - \hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+2} - \hat{\tau}_{L2jk}}\right)\right] \\ & \times \sin\{\omega_{IF,L2} t_i - [\hat{\phi}_{L2jk} \\ & + \hat{\omega}_{Dopp,L2jk}(t_i - \hat{\tau}_{L2jk})]\} \end{aligned} \tag{9}$$

where i_k is the index of the first RF front end sample time that obeys $\hat{\tau}_{L2jk} \le t_{i_k}$ and $N_k + 1$ is the total

number of samples that obey $\hat{\tau}_{L2jk} \leq t_i < \hat{\tau}_{L2jk+2}$. The L2 CL code accumulations are formed by replacing $C_{L2M0jk}[t]$ with $C_{L20Ljk}[t]$ in equations (8) and (9). Note how equations (8) and (9) sum over a nominal accumulation interval of 2 ms as opposed to 1 ms for the C/A code accumulations in equations (6) and (7). This is done purely for convenience. Note also that i_k and N_k in eqs. (8) and (9) will differ from i_k and N_k in eqs. (6) and (7).

IV. REVIEW OF BIT-WISE PARALLEL SOFTWARE CORRELATION

Base-Band Mixing

In order for a software receiver to efficiently perform base-band mixing, as in eqs. (8) and (9), the sine and cosine replicas must be pre-computed and stored in memory. It is desirable to minimize the number of sine and cosine signals that are stored in memory. The method implemented in this receiver was first demonstrated in [3]. This reference demonstrates a base-band mixing method which involves storing sine and cosine signals on a rough frequency grid. The method mixes the input RF signal with the carrier replicas that are closest in frequency to the estimated Doppler shift. The signals have an initial phase offset of zero. The resultant I and Q accumulations are rotated in such a way that they mimic mixing by carrier replicas with the appropriate frequency and phase. The resulting L1 C/A code accumulations are

$$\begin{aligned} I_{gL1jk}(\Delta) = & \sum_{i=i_k}^{i_k+N_k} y(t_i) \\ & \times C_{L1j}\left[0.001\left(\frac{t_i+\Delta-\hat{\tau}_{L1jk}}{\hat{\tau}_{L1jk+1}-\hat{\tau}_{L1jk}}\right)\right] \\ & \times \cos[(\omega_{IF,L1}+\omega_{gL1jk})(t_i-t_{0gL1jk})] \end{aligned} \tag{10}$$

$$\begin{aligned} Q_{gL1jk}(\Delta) = & \sum_{i=i_k}^{i_k+N_k} y(t_i) \\ & \times C_{L1j}\left[0.001\left(\frac{t_i+\Delta-\hat{\tau}_{L2jk}}{\hat{\tau}_{L1jk+1}-\hat{\tau}_{L1jk}}\right)\right] \\ & \times \sin[(\omega_{IF,L1}+\omega_{gL1jk})(t_i-t_{0gL1jk})] \end{aligned} \tag{11}$$

where ω_{L1gjk} is the grid Doppler shift frequency that is closest to the estimated frequency $\hat{\omega}_{Dopp,L1jk}$ and where t_{0gL1jk} is the time at which this carrier replica has zero carrier phase. These accumulations are then rotated in order to create accurate approximations of what would have been computed had the estimated carrier phase time history in equations (6) and (7) been used:

$$\begin{aligned} I_{L1jk}(\Delta) = & I_{gL1jk}(\Delta)\cos(\Delta\phi_{L1avgjk}) \\ & +Q_{gL1jk}(\Delta)\sin(\Delta\phi_{L1avgjk}) \end{aligned} \tag{12}$$

$$\begin{aligned} Q_{L1jk}(\Delta) = & -I_{gL1jk}(\Delta)\sin(\Delta\phi_{L1avgjk}) \\ & +Q_{gL1jk}(\Delta)\cos(\Delta\phi_{L1avgjk}) \end{aligned} \tag{13}$$

where $\Delta\phi_{L1avgjk}$ is the average phase difference between the grid carrier phase and the estimated carrier phase averaged over the accumulation interval:

$$\begin{aligned} \Delta\phi_{L1avgjk} = & \omega_{gL1jk}\left(\frac{\hat{\tau}_{L1jk}+\hat{\tau}_{L1jk+1}}{2}-t_{0gL1jk}\right) \\ & -\hat{\phi}_{L1jk}-\hat{\omega}_{Dopp,L1jk}\left(\frac{\hat{\tau}_{L1jk+1}-\hat{\tau}_{L1jk}}{2}\right) \\ & -\omega_{IF,L1}t_{0gL1jk}. \end{aligned} \tag{14}$$

The correlation of the time-division multiplexed CM and CL codes is complicated by the fact that navigation data is modulated on the CM code. This requires a modification to the code replicas used for correlation. Two CM/CL code replicas are generated. One replica has a +1 data bit on the CM code and is denoted the +CM/CL replica. The other one has a –1 bit on the CM code and is denoted the –CM/CL replica. The accumulations from the two replicas can produce either the CM code accumulation via subtraction or the CL code accumulation via addition.

The resulting L2 CM/CL code accumulations are

$$\begin{aligned} I_{gL2plusjk}(\Delta) = & \sum_{i=i_k}^{i_k+N_k} y(t_i)C_{L2plusjk}\Bigg[0.002 \\ & \times\left(\frac{t_i+\Delta-\hat{\tau}_{L1jk}}{\hat{\tau}_{L2jk+2}-\hat{\tau}_{L2jk}}\right)\Bigg] \\ & \times\cos[(\omega_{IF,L2}-\omega_{gL2jk}) \\ & \times(t_i-t_{0gL2jk})] \end{aligned} \tag{15}$$

$$\begin{aligned} Q_{gL2plusjk}(\Delta) = & -\sum_{i=i_k}^{i_k+N_k} y(t_i)C_{L2plusjk}\Bigg[0.002 \\ & \times\left(\frac{t_i+\Delta-\hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+2}-\hat{\tau}_{L2jk}}\right)\Bigg] \\ & \times\sin[(\omega_{IF,L2}-\omega_{gL2jk}) \\ & \times(t_i-t_{0gL2jk})] \end{aligned} \tag{16}$$

$$I_{gL2minusjk}(\Delta) = \sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L2minusjk} \left[0.002 \times \left(\frac{t_i + \Delta - \hat{\tau}_{L1jk}}{\hat{\tau}_{L2jk+2} - \hat{\tau}_{L2jk}} \right) \right] \times \cos[(\omega_{IF,L2} - \omega_{gL2jk}) \times (t_i - t_{0gL2jk})] \tag{17}$$

$$Q_{gL2minusjk}(\Delta) = -\sum_{i=i_k}^{i_k+N_k} y(t_i) C_{L2minusjk} \left[0.002 \times \left(\frac{t_i + \Delta - \hat{\tau}_{L2jk}}{\hat{\tau}_{L2jk+2} - \hat{\tau}_{L2jk}} \right) \right] \times \sin[(\omega_{IF,L2} - \omega_{gL2jk}) \times (t_i - t_{0gL2jk})] \tag{18}$$

where $C_{L2plusjk}$ is a portion of the over-sampled CM/CL code with a +1 data bit on the CM code, $C_{L2minusjk}$ is a portion of the over-sampled CM/CL code with a –1 data bit on the CM code, ω_{L2gjk} is the grid Doppler shift frequency that is closest to the estimated frequency $\hat{\omega}_{Dopp,L2jk}$, and t_{0gL2jk} is the time at which this carrier replica has zero carrier phase.

The in-phase CM code accumulation is formed by subtracting the accumulations in eqs. (15) and (17). The quadrature accumulation is formed by subtracting eqs. (16) and (18):

$$I_{gL2M0jk}(\Delta) = [I_{gL2plusjk}(\Delta) - I_{gL2minusjk}(\Delta)]/2 \tag{19}$$

$$Q_{gL2M0jk}(\Delta) = [Q_{gL2plusjk}(\Delta) - Q_{gL2minusjk}(\Delta)]/2 \tag{20}$$

The in-phase CL code accumulation is formed by adding the accumulations in equations (15) and (17). The quadrature accumulation is formed by adding equations (16) and (18):

$$I_{gL20Ljk}(\Delta) = [I_{gL2plusjk}(\Delta) + I_{gL2minusjk}(\Delta)]/2 \tag{21}$$

$$Q_{gL20Ljk}(\Delta) = [Q_{gL2plusjk}(\Delta) + Q_{gL2minusjk}(\Delta)]/2 \tag{22}$$

These accumulations are then rotated in order to create accurate approximations of what would have been computed had the estimated carrier phase time history in equations (8) and (9) been used. The CM code accumulations are

$$I_{L2M0jk}(\Delta) = I_{gL2M0jk}(\Delta)\cos(\Delta\phi_{L2avgjk}) + Q_{gL2M0jk}(\Delta)\sin(\Delta\phi_{L2avgjk}) \tag{23}$$

$$Q_{L2M0jk}(\Delta) = -I_{gL2M0jk}(\Delta)\sin(\Delta\phi_{L2avgjk}) + Q_{gL2M0jk}(\Delta)\cos(\Delta\phi_{L2avgjk}) \tag{24}$$

and the CL code accumulations are

$$I_{L20Ljk}(\Delta) = I_{gL20Ljk}(\Delta)\cos(\Delta\phi_{L2avgjk}) + Q_{gL20Ljk}(\Delta)\sin(\Delta\phi_{L2avgjk}) \tag{25}$$

$$Q_{L20Ljk}(\Delta) = -I_{gL20Ljk}(\Delta)\sin(\Delta\phi_{L2avgjk}) + Q_{gL20Ljk}(\Delta)\cos(\Delta\phi_{L2avgjk}) \tag{26}$$

where $\Delta\phi_{L2avgjk}$ is the average phase difference between the grid carrier phase and the estimated carrier phase averaged over the accumulation interval:

$$\Delta\phi_{L2avgjk} = \omega_{gL2jk}\left(\frac{\hat{\tau}_{L2jk} + \hat{\tau}_{L2jk+2}}{2} - t_{0gL2jk} \right) - \hat{\phi}_{L2jk} - \hat{\omega}_{Dopp,L2jk} \times \left(\frac{\hat{\tau}_{L2jk+2} - \hat{\tau}_{L2jk}}{2} \right) + \omega_{IF,L2} t_{0gL2jk}. \tag{27}$$

A slight loss in C/N_0 occurs when the grid frequency is not the true Doppler shift.The worst-case loss function is expressed as a function of the frequency grid spacing Δf and is given by

$$\Delta SNR = 20\log_{10}\left[\frac{\sin(\pi\Delta f T/2)}{\pi\Delta f T/2} \right] \tag{28}$$

where Δf is in units of Hz, and T is the integration period in sec. With Δf equal to 175 Hz, the worst-case SNR loss is 0.11 dB for the C/A code. The worst-case SNR loss for the CM/CL code is 0.11 dB if T=0.002 and Δf=87.5 Hz. The use of a closer frequency spacing for the grid of L2 Doppler shifts counteracts the use of a longer accumulation interval for the L2 codes.

The nominal accumulation interval of 0.002 seconds for the CM/CL code is used for programming convenience,

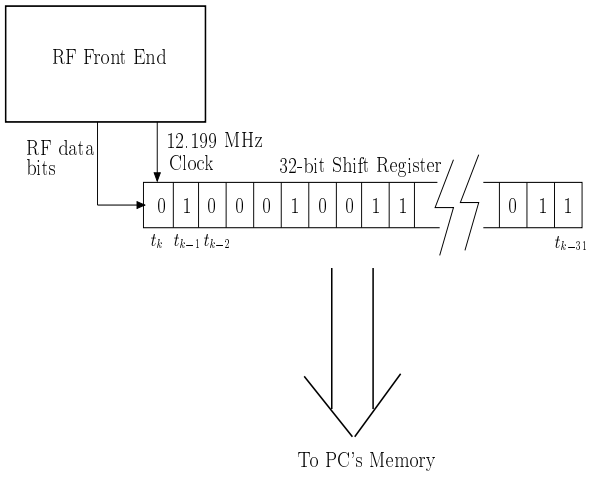

Fig. 3. Schematic diagram of how the RF front end's 1-bit samples are bit-packed into a shift register prior to being read into memory.

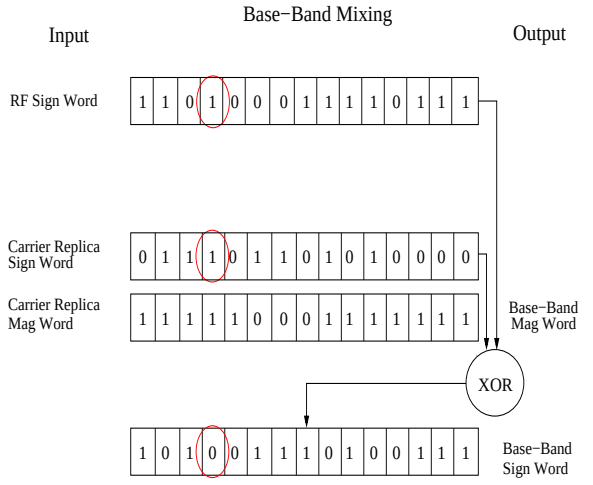

Fig. 4. Illustrative example of bit-wise parallel base-band mixing of a 16-bit RF data sign word and a 16-bit carrier replica with magnitude and sign words.

but it can be changed. It is clear from eq. (28) that lengthening the accumulation interval causes a decrease in SNR due to accumulated phase differences of the received signal and the carrier replica. Decreasing the frequency grid step size decreases this loss, but increases the required memory for storing the receiver's required ±10 KHz range of carrier replicas. Therefore, it has been decided not to lengthen this interval.

The cosine and sine signals on the Doppler frequency grid are stored as 2-bit binary sign and magnitude representations. These carrier replicas are stored in a bit-packed fashion such that successive samples of a replica signal are stored in successive bits in 32-bit words. Each tabulated carrier replica's sign bits are stored in a sequence of 32-bit sign words, and the magnitude bits are stored in a sequence of 32-bit magnitude words. This format is the same as in [3], [4].

Recall from the System Configuration section that 32 successive 1-bit RF data samples are bit-packed into a 32-bit shift register prior to being read into memory. Fig. 3 shows the RF front end, the RF front end sign bits, and the shift register. The sign bits are input into the shift register at the sampling rate of 12.199 MHz. When the 32-bit shift register is full, its contents are read into memory. The 32-bit words, once in memory, are called the RF data sign words. Successive words are stored in a circular buffer for processing by the software correlator. Bit-packing of the RF data samples motivates the use of bit-wise parallel correlation and accumulation of the GPS signals.

Bit-wise parallel base-band mixing makes use of the EXCLUSIVE OR operation and the fact that the RF data stream is bit-packed into words prior to being read into memory. The base-band signal's sign words are computed by taking the EXCLUSIVE OR of the RF data sign words and the carrier replica sign words. The base-band signal's magnitude words are simply the transcripted carrier replica magnitude words. This procedure is repeated for all the words that constitute an accumulation interval. The resultant base-band signal has a 2-bit representation with possible values of ±1 and ±2. Fig. 4 shows an example of bit-wise parallel base-band mixing of a 16-bit RF data sign word and 16-bit carrier replica sign and magnitude words. 16-bit words are used in place of this receiver's 32-bit words solely for illustrative purposes. Fig. 4 shows how a single EXCLUSIVE OR operation of the RF data sign word and carrier replica sign word produces the base-band sign word. The carrier replica magnitude word is simply renamed the base-band magnitude word. A thorough explanation of bit-wise parallel base-band mixing can be found in [3], [4].

Mixing of the Base-Band Signal with a Local PRN Code

Both prompt and early-minus-late correlations are needed to track the carrier frequency, carrier phase, and code phase in a GPS receiver. The L1 C/A code and L2 CM code prompt correlations are defined by the equation pairs (6), (7) and (8), (9) with $\Delta = 0$. The early-minus-late correlations for the L1 C/A code are $I_{L1jk}(\Delta t_{eml}/2) - I_{L1jk}(-\Delta t_{eml}/2)$ and $Q_{L1jk}(\Delta t_{eml}/2) - Q_{L1jk}(-\Delta t_{eml}/2)$ and for the L2 CM code $I_{L2M0jk}(\Delta t_{eml}/2) - I_{L2M0jk}(-\Delta t_{eml}/2)$ and $Q_{L2M0jk}(\Delta t_{eml}/2) - Q_{L2M0jk}(-\Delta t_{eml}/2)$, where Δt_{eml} is the spacing between the early and late PRN code replicas. The L2 CL correlations are defined similarly

but with $C_{L2M0jk}[t]$ replaced by $C_{L20Ljk}[t]$ in equations (8) and (9).

The prompt and early-minus-late PRN code replicas can be mixed with the base-band mixed signal by bit redefinitions and a simple *EXCLUSIVE OR* operation of the two input signals' sign words. The mixing of the early-minus-late code with the base-band signal also requires a transcription of zero mask bits. These operations are described in [3], [4].

Bit-Wise Parallel Accumulation of Correlations

The final operation in the correlation calculations is to sum the results over all of the samples in a given estimated PRN code interval. This operation requires additional bit-wise parallel operations followed by operations that form totals over the bits in a given word.

The code-correlated base-band signals have a 2-bit representation. The possible values are ±1 and ±2. To sum this resultant signal in an efficient manner, the four different combinations of the 2-bit representation are represented by four 32-bit words. These four 32-bit words are called value words. The four value words are denoted the +1 value word, the –1 value word, the +2 value word, and the –2 value word. A value word contains a 1 in the bit location where the resultant signal's 2-bit representation matches the value word's corresponding value. For example, the +1 value word contains 1s in all the bit locations in which the resultant signal has a sign bit equal to 1, which indicates a positive sign, and a magnitude bit equal to 0, which indicates a magnitude of 1. Boolean logic operations are used to generate the four value words. [3] and [4] give a full description of this process.

The accumulation operation must sum the number of 1 bits in each of the four value words. The summations are accomplished using a table look-up. The value word is used as the address in the memory table, and the table's output is set up to deliver the number of 1 values in the address. A 16-bit table has been used. This gives it a memory size of 2^{16} bytes or 64 Kbytes, which makes it able to fit into the microprocessor's cache and allows for very fast execution. If this code is implemented on a DSP chip, then it can use a machine instruction for summing the 1 bits.

V. REAL-TIME GENERATION OF OVER-SAMPLED PRN CODES

The software correlator needs to generate bit-wise parallel representations of the over-sampled PRN codes used in eqs. (10), (11), and (15)–(18). The needed over-sampled PRN codes are $C_{L1j}[t]$, $C_{L2plusjk}[t]$, and $C_{L2minusjk}[t]$. The software correlator needs prompt and early-minus-late versions of these three codes, and it needs them in bit-wise parallel format. The software receiver described in [3] and [4] only needed the $C_{L1j}[t]$ code. It pre-computed a full 1 msec period of this code and made a table of pre-computed code period time histories for a range of code phases as measured with respect to the sample time. Such an approach is impractical in the present case because the $C_{L2plusjk}$ and $C_{L2minusjk}$ code segments have to be generated in 2 msec segments for 750 different possible values of k, 31 different possible values of j, and at multiple code phase offsets. Pre-computation would require an impractical amount of computer memory.

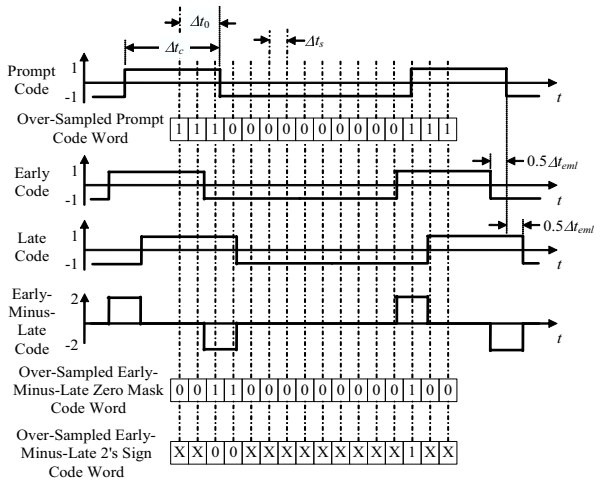

Fig. 5. Example sections of prompt, early, late, and early-minus-late PRN code time histories and 16-bit word representations of their over-sampled equivalents.

Over-Sampling and Bit-Wise Parallel Storage of PRN Codes

This section describes an efficient method for real-time generation of bit-wise parallel representations of the over-sampled versions of the required PRN codes. The C_{L1j}, $C_{L2plusjk}$, and $C_{L2minusjk}$ PRN codes are sequences of +1 and –1 chip values. Suppose that the receiver's code replica progresses through its values at the chipping rate $f_c = 1/\Delta t_c$, where Δt_c is the chip length. Suppose, also, that the receiver's early chips start and stop $0.5\Delta t_{eml}$ seconds before the prompt chips and that its late chips start and stop $0.5\Delta t_{eml}$ seconds after the prompt chips. Fig. 5 depicts example segments of the prompt, early, and late code replicas of a typical PRN code. The early-minus-late code, which equals the difference between the early and late codes, is also shown in Fig. 5.

The receiver processes a raw RF input signal that is sampled at the rate $f_s = 1/\Delta t_s$ Hz, where $\Delta t_s = t_{i+1} - t_i$. Recall that $f_s = 12.199$ MHz for the present receiver. It also needs prompt and early-minus-late code

replicas sampled at the same times as the raw RF data. Sixteen RF sample times are depicted in Fig. 5 as vertical dash-dotted lines. The receiver's sampled versions of the PRN codes are called over-sampled because $\Delta t_s < \Delta t_c$, as shown in Fig. 5, which implies that $f_s > f_c$.

The sampled values of the prompt and early-minus-late PRN codes can be represented in a bit-wise parallel format as integer words. The prompt code can be represented by its sign bit at each sample time, with a 1 bit representing +1 and a 0 bit representing –1. The bit-wise parallel representation of the prompt code at the 16 sample times of Fig. 5 is shown immediately below the prompt code's time history plot. It starts with three 1s, continues with ten 0s, and finishes with another three 1s. It is stored as the integer $2^{15} + 2^{14} + 2^{13} + 2^2 + 2^1 + 2^0 = 57351$.

The early-minus-late code requires a 1.5-bit representation. A zero-mask bit is 0 if the sampled early-minus-late code equals 0, and it is 1 if the early-minus-late code equals +2 or –2. The parallel representation of the zero-mask bits at the 16 sample times of Fig. 5 appears immediately below the early-minus-late code plot. Its integer equivalent is $2^{13}+2^{12}+2^2 = 12292$. The representation's 2's sign bit equals 1 if the early-minus-late code equals +2, and it equals 0 if the code equals –2. The 2's sign bit is irrelevant if the corresponding zero-mask bit equals zero. The bit-wise parallel representation of the 2's sign bits for the 16 sample times on Fig. 5 is shown below the zero-mask representation. Its X values indicate bit values that are irrelevant because the corresponding zero-mask bits equal 0.

Table Look-Ups of Bit-Wise Parallel Representations of Over-Sampled Codes

Table look-ups are used to translate from a PRN code and its timing information to bit-wise parallel representations of its over-sampled prompt and early-minus-late versions. The table designs exploit the constancy of the sampling rate, the nominal chipping rate, the receiver's nominal early-to-late code delay, and the known maximum number of chips that span a data word. The tables have 2 variable inputs: the sequence of code chip values and the end time of the initial prompt chip relative to the data word's first sample, Δt_0. A separate table is used for each of the three output code words defined in Fig. 5, the prompt sign word, the early-minus-late zero-mask word, and the early-minus-late 2's sign word.

It is an approximation to assume that the chipping rate equals its nominal value. It usually includes a Doppler shift. This approximation does not cause significant code distortion if the data word duration is short and if one corrects Δt_0 for the average effects of the Doppler shift.

The time offset input Δt_0 can take on any value in the continuous range:

$$-\frac{1}{2}\Delta t_{eml} < \Delta t_0 \le \Delta t_c - \frac{1}{2}\Delta t_{eml} \tag{29}$$

This range's lower limit guarantees that the end time of the first late chip occurs no earlier than the first sample time. The upper limit guarantees that the start time of the first late chip occurs no later than the first sample.

This continuous range of possible Δt_0 values is replaced with a discrete grid in order to create a practical table. The grid spacing is $\Delta t_s/m$. The integer m is chosen to be large enough to guarantee sufficient PRN code timing resolution. In GPS applications, one typically chooses m so that $(c\Delta t_s/m)$ is on the order of several meters or less, where c is the speed of light. One avoids a very large m because the table sizes are proportional to m. In the present receiver m has been chosen to be 14. Given m, the grid of relative end times of the first prompt code period is:

$$\Delta t_{0k} = \frac{k\Delta t_s}{m} \quad \text{for } k = k_{min}, ..., k_{max} \tag{30}$$

where the limits

$$k_{min} = floor\left(-\frac{m\Delta t_{eml}}{2\Delta t_s}\right) - 2 \tag{31a}$$

$$k_{max} = floor\left[\frac{m(\Delta t_c - 0.5\Delta t_{eml})}{\Delta t_s}\right] \tag{31b}$$

provide full coverage of the interval defined in eq. (29). The $floor()$ function rounds to the nearest integer in the direction of $-\infty$. These limits are $k_{min} = -44$ and $k_{max} = 125$ for the present receiver.

Table construction requires knowledge of the number of code chips needed to ensure that the prompt, early, and late code versions all span the entire data word. Given Δt_0, chip values for

$$l(\Delta t_0) = floor\left[\frac{(n_s - 1)\Delta t_s - \Delta t_0 + 0.5\Delta t_{eml}}{\Delta t_c}\right] + 2 \tag{32}$$

code chips are needed in order to determine the prompt, early, and late codes at all of the n_s sample times of a data word. $l(\Delta t_0)$ is a non-increasing function of Δt_0. Therefore, the following gives the maximum number of chips that are needed in order to cover a full data word:

$$L = l(\Delta t_{0kmin}) \tag{33}$$

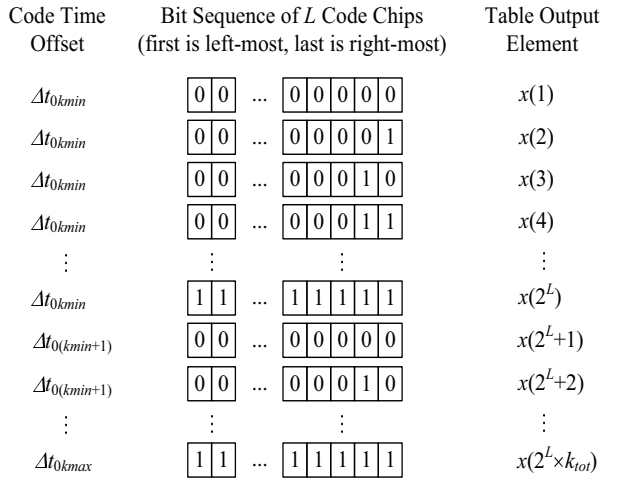

Fig. 6. Layout of code over-sampling table as a function of code time offset and chip sequence.

The present receiver uses $n_s = 32$ samples per data word, and its maximum number of code chips per data word is $L = 5$.

The size of each table can be determined from the parameters k_{min}, k_{max}, and L. The time grid contains $k_{tot} = (k_{max} - k_{min} + 1)$ different offsets of the first code chip, and there are 2^L possible chip sequences. Thus, each table contains $k_{tot} \times 2^L$ entries, and each entry is an unsigned integer in the range 0 to $2^{n_s} - 1$. The number of entries in each table is 5440 for the present receiver.

Each table is stored as an array with a single index. The first 2^L entries tabulate the 2^L different possible chip sequences when $\Delta t_0 = \Delta t_{0kmin}$, the next 2^L entries correspond to $\Delta t_0 = \Delta t_{0(kmin+1)}$, and so forth. The tabulated bit sequences for a fixed Δt_0 are ordered by interpreting the sequence as a binary index counter with the first chip being the most significant counter bit and the L^{th} chip being the least significant bit. This table layout is depicted in Fig. 6. The integer elements of the table output are the $x(i)$ values that are listed in the right-hand column.

The table index can be computed from the time offset index k and the code bit sequence. Suppose that the code chip sequence is $C(1)$, $C(2)$, $C(3)$, ..., $C(L)$, where $C(j) = 0$ represents a –1 code chip and $C(j) = 1$ represents a +1 chip. Then the Fig. 6 table index is:

$$i[k, C(1), C(2), C(3), ..., C(L)] = 1 + (k - k_{min}) \times 2^L + \sum_{j=1}^{L} C(j) 2^{L-j} \quad (34)$$

This equation can be inverted to give k and the chip sequence as functions of the index i:

$$k(i) = k_{min} + floor\left(\frac{i-1}{2^L}\right) \quad (35a)$$

$$C[j;i] = mod\left(floor\left\{\frac{mod[(i-1), 2^L]}{2^{L-j}}\right\}, 2\right) \quad \text{for } j = 1, 2, 3, ..., L \quad (35b)$$

where $mod(y, z) = y - z \times floor(y/z)$ is the usual remainder function.

The following computations generate the $x(i)$ entries of the 3 tables. Given i, the time offset index $k(i)$ is computed from eq. (35a) and is used to generate 3 sequences of chip indices:

$$j_p(n,i) = 2 + floor\left\{\left[n - 1 - \frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right]\right\} \quad \text{for } n = 1, 2, 3, ..., n_s \quad (36a)$$

$$j_e(n,i) = 2 + floor\left\{\left[n - 1 - \frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right] + \left[\frac{\Delta t_{eml}}{2\Delta t_c}\right]\right\} \quad \text{for } n = 1, 2, 3, ..., n_s \quad (36b)$$

$$j_l(n,i) = 2 + floor\left\{\left[n - 1 - \frac{k(i)}{m}\right]\left[\frac{\Delta t_s}{\Delta t_c}\right] - \left[\frac{\Delta t_{eml}}{2\Delta t_c}\right]\right\} \quad \text{for } n = 1, 2, 3, ..., n_s \quad (36c)$$

where n is the index of the sample time within the over-sampled data word. The index $j_p(n,i)$ identifies the PRN code chip that applies at sample n for the prompt code. The indices $j_e(n,i)$ and $j_l(n,i)$ are defined similarly for the early and late codes, respectively. These indices, in turn, are used in eq. (35b) to determine the chip values that apply at the sample times:

$$C_p(n,i) = C[j_p(n,i); i] \quad \text{for } n = 1, 2, 3, ..., n_s \quad (37a)$$

$$C_e(n,i) = C[j_e(n,i); i] \quad \text{for } n = 1, 2, 3, ..., n_s \quad (37b)$$

$$C_l(n,i) = C[j_l(n,i); i] \quad \text{for } n = 1, 2, 3, ..., n_s \quad (37c)$$

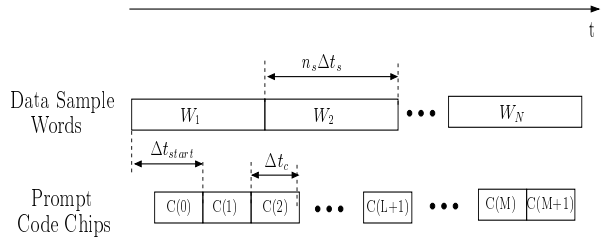

Fig. 7. Timing relationship between data sample words and the sequence of prompt code chips that defines an accumulation interval.

where $C_p(n,i)$, $C_e(n,i)$ and $C_l(n,i)$ are, respectively, the over-sampled prompt, early, and late code chip values. These values are used to generate the unsigned integers that constitute the 3 tables's bit-wise parallel code representations:

$$x_p(i) = \sum_{n=1}^{n_s} C_p(n,i) \times 2^{n_s-n} \tag{38a}$$

$$x_{emlzm}(i) = \sum_{n=1}^{n_s} mod\{[C_e(n,i) + C_l(n,i)], 2\} \times 2^{n_s-n} \tag{38b}$$

$$x_{eml2s}(i) = \sum_{n=1}^{n_s} mod\{[C_e(n,i) + C_l(n,i)], 2\} \times C_e(n,i) \times 2^{n_s-n} \tag{38c}$$

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

Efficient Table Look-Ups of Over-Sampled Codes During an Accumulation

Delineation of an Accumulation Interval in Terms of a Prompt Code Chip Sequence Each set of accumulation calculations in the software correlator works with a fixed sequence of code chips. The prompt code has a specified timing relationship to the incoming RF data stream, as shown in Fig. 7. The accumulation interval starts when prompt chip $C(1)$ starts and ends when prompt chip $C(M)$ ends. The receiver uses $M = 1023$ chips when calculating the 1 msec L1 C/A accumulations in eqs. (10) and (11), and it uses $M = 2046$ chips when calculating the 2 msec +CM/CL and –CM/CL accumulations in eqs. (15)–(18). The chip sequence starts Δt_{start} seconds past the first sample of data word W_1, it chips at the constant rate $f_c = 1/\Delta t_c$, and it ends $(\Delta t_{start} + M\Delta t_c)$ seconds after the first sample of data word W_1. Prompt chip $C(M)$ ends during data word W_N, which implies that

$$N = ceil\left(\frac{\Delta t_{start} + M\Delta t_c}{n_s \Delta t_s}\right) \tag{39}$$

where the $ceil()$ function rounds to the nearest integer towards $+\infty$.

Some of the initial bits of word W_1 and some of the final bits of word W_N may not be included in the accumulation. Let n_{ex0} be the number of initial bits of W_1 that are excluded, and let n_{exf} be the number of final bits of W_N that are excluded. Fig. 7 implies that

$$n_{ex0} = ceil\left(\frac{\Delta t_{start}}{\Delta t_s}\right) \tag{40a}$$

$$n_{exf} = n_s N - ceil\left(\frac{\Delta t_{start} + M\Delta t_c}{\Delta t_s}\right) \tag{40b}$$

These numbers define additional zero-mask words that the receiver uses to properly process the first and last data words during its bit-wise parallel accumulation calculations.

The remainder of this sub-section explains how to efficiently determine the correct $x_p(i)$, $x_{emlzm}(i)$, and $x_{eml2s}(i)$ bit-wise parallel code representations for the N data words W_1 through W_N. This amounts to making an efficient determination of the correct table index i_ν that corresponds to data word W_ν for $\nu = 1, ..., N$.

Generation of an Auxiliary Table of Index Components from the PRN Code's Bit Sequence The first step of the index calculation is to compute an auxiliary table of candidate integers for the final summation term that appears on the right-hand side of eq. (34):

$$\Delta i(\mu) = \sum_{j=1}^{L} C(\mu + j - L - 1)2^{L-j} \quad \text{for } \mu = 1, 2, 3, ..., (M + L + 1) \tag{41}$$

This computation assumes that $C(0)$ through $C(M+1)$ are known from the outputs of software feedback shift registers that generate the C_{L1j}, $C_{L2plusjk}$, or $C_{L2minusjk}$ code as described in [10] and in Section IV. It uses 0 values for $C(-L+1)$, $C(-L+2)$, $C(-L+3)$, ..., $C(-1)$ and for $C(M+2)$, $C(M+3)$, $C(M+4)$, ..., $C(M+L)$. These 0 chips are place holders that affect the over-sampled codes only for the first n_{ex0} samples of data word W_1 and the last n_{exf} samples of data word W_N. The following iteration constructs the auxiliary table:

$$\Delta i(1) = C(0) \tag{42a}$$

$$\Delta i(\mu) = \begin{array}{l} mod[2\Delta i(\mu-1), 2^L] + C(\mu-1) \\ \text{for } \mu = 2,3,4,...,(M+2) \end{array} \quad (42b)$$

$$\Delta i(\mu) = \begin{array}{l} mod[2\Delta i(\mu-1), 2^L] \\ \text{for } \mu = (M+3),(M+4),...,(M+L+1) \end{array} \quad (42c)$$

Note that the $mod(2\times, 2^L)$ operation in the latter 2 equations can be replaced by a truncated leftward bit shift, which is available as a single instruction on many processors.

Iterative Calculation of Table Indices Determination of the correct index into the $x_p(i)$, $x_{emlzm}(i)$, and $x_{eml2s}(i)$ tables for data word W_ν can be reduced to the determination of two quantities. One is the time offset index k_ν that causes $\Delta t_{0k\nu}$ from eq. (30) to match the true time offset for data word W_ν as closely as possible. The other quantity is the auxiliary table index μ_ν. It defines the sequence of code chips that are associated with data word W_ν. Given these quantities, the correct index for the three $x(i)$ tables is

$$i_\nu = 1 + (k_\nu - k_{min}) \times 2^L + \Delta i(\mu_\nu) \text{ for } \nu = 1,2,3,...,N \quad (43)$$

The determination of k_ν and μ_ν is accomplished via timing calculations. The auxiliary index μ_ν is a function of the position of the W_ν data word relative to the PRN code sequence. The index k_ν is calculated from the position of the first code chip that is associated with μ_ν.

An integer measure of time is used. It allows the μ_ν and k_ν computations to be carried out using efficient integer operations. Integer time is measured in fine-scale time units of length:

$$\Delta t_f = \frac{\Delta t_s}{m_f} \quad (44)$$

where m_f is the integer number of fine-scale time intervals per sample interval. This number is chosen large enough to preclude any significant build-up of timing errors during an accumulation interval due to the finite time resolution Δt_f. A good rule of thumb is to choose $m_f > 2mN$. Recall that $\Delta t_s/m$ is the timing resolution of the $x(i)$ tables and that N is the number of data words in the accumulation interval. The calculation of the k_ν values over one accumulation interval involves approximately N iterative time increments, each of which has a resolution of Δt_f. If m_f obeys the inequality given above, then the timing errors due to the finite precision Δt_f will be less than the timing error caused by the finite timing precision of the $x(i)$ tables. Normally, it is possible to make m_f much larger than $2mN$ and still keep all of the relevant indexing calculations within the size limits of a 32-bit signed integer. One should choose m_f to be a power of 2 so that a rightward bit shift operation can be used to implement integer division by m_f. The value of m_f used in the present receiver is 2^{20}.

The new fine-scale time unit can be used to define an integer that approximately keeps track of the code time offset $\Delta t_{0\nu}$ for data word W_ν:

$$k_{f\nu} \cong round\Big(\frac{\Delta t_{0\nu}}{\Delta t_f}\Big) = round\Big(\frac{m_f \Delta t_{0\nu}}{\Delta t_s}\Big) \quad (45)$$

where the $round()$ function rounds up or down to the nearest integer. The interval $\Delta t_{0\nu}$ is the lag of the end time of chip $C(\mu_\nu - L)$ behind the first sample time of data word W_ν – review Fig. 7. The algorithm that iteratively determines $k_{f\nu}$ tries to keep the relationship in eq. (45) exact, but the integer timing calculations allow small errors to build up. Note that $k_{f\nu}/m_f \cong k_\nu/m$, as implied by a comparison of eqs. (30) and (45). This relationship will be used to determine k_ν from $k_{f\nu}$.

Several constants are required by the iterative procedure that determines $k_{f\nu}$, k_ν, and μ_ν. The first five constants account for the difference between the nominal chip length used to generate the $x(i)$ tables, Δt_{cnom}, and the actual chip length used in the accumulation, Δt_c:

$$k_{fmid} = round\left[\frac{(n_s - 1)m_f}{2}\right] \quad (46a)$$

$$\lambda = \frac{\Delta t_c - \Delta t_{cnom}}{\Delta t_c} \quad (46b)$$

$$a_{fix0} = ceil\left[\left(k_{fmid} - \frac{m_f k_{min}}{m}\right)\lambda^2\right] sign(\lambda) \quad (46c)$$

$$b_{fix} = \begin{cases} 1 & \text{if } \Delta t_c = \Delta t_{cnom} \\ 2^{ceil[\ln(a_{fix0}/\lambda)/\ln(2)]} & \text{if } \Delta t_c \neq \Delta t_{cnom} \end{cases} \quad (46d)$$

$$a_{fix} = round(\lambda b_{fix}) \quad (46e)$$

where the $sign()$ function returns +1 if its input argument is positive, 0 if the argument is 0, and –1 if the argument is negative. The index k_{fmid} is approximately half the length of a data word as measured in Δt_f time units. It gets used in conjunction with the rational factor a_{fix}/b_{fix} to compute a corrected $k_{f\nu}$ value that removes the average effect of the difference between Δt_{cnom} and Δt_c:

$$k_{f\nu fix}(k_{f\nu}) = k_{f\nu} + round\left[\left(k_{fmid} - k_{f\nu}\right)\frac{a_{fix}}{b_{fix}}\right] \quad (47)$$

Equation (46d) picks b_{fix} to equal a power of 2 so that the integer division by b_{fix} in eq. (47) can be accomplished using a rightward bit shift operation. The $round()$ operation in eq. (47) can be accomplished as part of the division if one first adds $sign(a_{fix}) \times b_{fix}/2$ to the quantity $(k_{fmid} - k_{f\nu}) \times a_{fix}$ before performing the rightward bit shift that constitutes division by b_{fix}. Note that this round implementation presumes that $k_{fmid} > k_{f\nu}$, which is true in most applications.

Five additional constants help to define the $k_{f\nu}$ and μ_ν iterations:

$$L_{typ} = round\left(\frac{n_s \Delta t_s}{\Delta t_c}\right) \quad (48a)$$

$$\Delta k_{fc} = round\left(\frac{m_f \Delta t_c}{\Delta t_s}\right) \quad (48b)$$

$$\Delta k_{ftyp} = round\left(\frac{m_f L_{typ} \Delta t_c}{\Delta t_s}\right) - n_s m_f \quad (48c)$$

$$k_{fmin} = round\left(\left\{\left[\frac{m_f(k_{min}+1)}{m}\right] - \left[\frac{a_{fix}k_{fmid}}{b_{fix}}\right]\right\} \div \left[1 - \frac{a_{fix}}{b_{fix}}\right]\right) \quad (48d)$$

$$k_{fmax} = round\left(\left\{\left[\frac{m_f(k_{max}-1)}{m}\right] - \left[\frac{a_{fix}k_{fmid}}{b_{fix}}\right]\right\} \div \left[1 - \frac{a_{fix}}{b_{fix}}\right]\right) \quad (48e)$$

The constant L_{typ} is the nominal number of code chips per data word. The constant Δk_{fc} equals the number of fine-scale time intervals per code chip. The constant Δk_{ftyp} is the nominal increment to $k_{f\nu}$ per data word. The limits k_{fmin} and k_{fmax} are approximately the limits k_{min} and k_{max} from eqs. (31a) and (31b) re-scaled to the new fine time scale and adjusted for the difference between the nominal code chipping rate of the $x(i)$ tables and the actual chipping rate of the accumulation. Note that the extra –2 on the right-hand side of eq. (31a) is compensated by the increment to k_{min} on the right-hand side of eq. (48d) and the decrement to k_{max} on the right-hand side of eq. (48e). The original –2 term and the increment and decrement ensure that k_f values which respect the limits in eq. (48d) and (48e) will be transformed into k values that respect the limits in eqs. (31a) and (31b).

The joint $k_{f\nu}$ and μ_ν iteration begins by computing nominal values for the first data word:

$$k_{f1nom} = round\left\{\left[\left(\frac{\Delta t_{start}}{\Delta t_c}\right) + 1 + floor\left(\frac{-\Delta t_{start}}{\Delta t_c}\right)\right]\left[\frac{\Delta t_c m_f}{\Delta t_s}\right]\right\} \quad (49a)$$

$$\mu_{1nom} = floor\left(\frac{-\Delta t_{start}}{\Delta t_c}\right) + 1 + L \quad (49b)$$

It is possible that k_{f1nom} from eq. (49a) will violate its upper limit k_{fmax}. Therefore, the following conditional adjustment must be implemented in order to finish the initialization.

$$k_{f1} = \begin{cases} k_{f1nom} & \text{if } k_{f1nom} \le k_{fmax} \\ k_{f1nom} - \Delta k_{fc} & \text{if } k_{fmax} < k_{f1nom} \end{cases} \quad (50a)$$

$$\mu_1 = \begin{cases} \mu_{1nom} & \text{if } k_{f1nom} \le k_{fmax} \\ \mu_{1nom} - 1 & \text{if } k_{fmax} < k_{f1nom} \end{cases} \quad (50b)$$

The calculation of (k_{f2}, μ_2), $(k_{f3}, \mu_3), (k_{f4}, \mu_4)$, ..., (k_{fN}, μ_N) proceeds according to the iteration:

$$k_{f\nu nom} = k_{f(\nu-1)} + \Delta k_{ftyp} \text{ for } \nu = 2, 3, 4, ..., N \quad (51a)$$

$$\mu_{\nu nom} = \mu_{(\nu-1)} + L_{typ} \text{ for } \nu = 2, 3, 4, ..., N \quad (51b)$$

$$k_{f\nu} = \begin{cases} k_{f\nu nom} + \Delta k_{fc} & \text{if } k_{f\nu nom} < k_{fmin} \\ k_{f\nu nom} & \text{if } k_{fmin} \le k_{f\nu nom} \le k_{fmax} \\ k_{f\nu nom} - \Delta k_{fc} & \text{if } k_{fmax} < k_{f\nu nom} \end{cases} \text{for } \nu = 2, 3, 4, ..., N \quad (52a)$$

$$\mu_\nu = \begin{cases} \mu_{\nu nom} + 1 & \text{if } k_{f\nu nom} < k_{fmin} \\ \mu_{\nu nom} & \text{if } k_{fmin} \le k_{f\nu nom} \le k_{fmax} \\ \mu_{\nu nom} - 1 & \text{if } k_{fmax} < k_{f\nu nom} \end{cases} \text{for } \nu = 2, 3, 4, ..., N \quad (52b)$$

Next, $k_{f\nu}$ is used to compute k_ν:

$$k_\nu = round\left[\frac{mk_{f\nu fix}(k_{f\nu})}{m_f}\right] \text{ for } \nu = 1,2,3,...,N \quad (53)$$

The $round()$ operation in eq. (53) can be implemented by adding $m_f/2$ to $m \times k_{f\nu fix}(k_{f\nu})$ before the rightward bit shift that constitutes division by m_f. The result of the division will be the correct value of k_ν for any sign of $k_{f\nu fix}(k_{f\nu})$ if the computer works with 2's compliment notation for signed integers and if the rightward bit shift fills in from the left with the 2's compliment sign bit.

The values k_ν from eq. (53) and μ_ν from eq. (52b) can be used in eq. (43) to compute i_ν, and i_ν can be used in the $x(i)$ tables to determine the over-sampled prompt sign, early-minus-late zero-mask, and early-minus-late 2's sign words, $x_{p\nu}$, $x_{emlzm\nu}$, and $x_{eml2s\nu}$:

$$x_{p\nu} = x_p(i_\nu) \text{ for } \nu = 1,2,3,...,N \quad (54a)$$

$$x_{emlzm\nu} = x_{emlzm}(i_\nu) \text{ for } \nu = 1,2,3,...,N \quad (54b)$$

$$x_{eml2s\nu} = x_{eml2s}(i_\nu) \text{ for } \nu = 1,2,3,...,N \quad (54c)$$

It is important to implement the computations in eqs. (51a)–(54c) efficiently because they get performed once per data word. One economy is to reduce the conditionals in eqs. (52a) and (52b) to a single conditional per data word during normal operation. This can be done because the sign of Δk_{ftyp} in eq. (51a) is fixed. If $\Delta k_{ftyp} < 0$, then one only needs to check whether $k_{f\nu nom} < k_{fmin}$. This is true because eq. (51a) only decrements $k_{f\nu}$ in this case. Conversely, if $\Delta k_{ftyp} > 0$, then one only needs to check whether $k_{f\nu nom} > k_{fmax}$. The decision about which condition to check can be made at the beginning of the accumulation because Δk_{ftyp} gets calculated prior to execution of the iteration in eqs. (51a)–(54c).

Additional efficiencies are important when using processors that create instruction pipelines. One must avoid "if" statements because they disrupt the pipeline. In this case, one replaces eqs. (52a) and (52b) with the following computations:

$$\eta_{f\nu} = \begin{cases} min[0, sign(k_{f\nu nom} - k_{fmin})] & \text{if } k_{ftyp} < 0 \\ 0 & \text{if } k_{ftyp} = 0 \\ max[0, sign(k_{f\nu nom} - k_{fmax})] & \text{if } k_{ftyp} > 0 \end{cases} \text{ for } \nu = 2,3,4,...,N \quad (55a)$$

$$k_{f\nu} = k_{f\nu nom} - \eta_{f\nu}\Delta k_{fc} \text{ for } \nu = 2,3,4,...,N \quad (55b)$$

$$\mu_\nu = \mu_{\nu nom} - \eta_{f\nu} \text{ for } \nu = 2,3,4,...,N \quad (55c)$$

The $min()$ and $max()$ functions return, respectively, the minimum or maximum of their two input arguments. The variable $\eta_{f\nu}$ is normally 0, in which case eqs. (55b) and (55c) leave $k_{f\nu nom}$ and $\mu_{\nu nom}$ unchanged. The value of $\eta_{f\nu}$ will be –1 if $\Delta k_{ftyp} < 0$ and $k_{f\nu nom} < k_{fmin}$, and it will be +1 if $\Delta k_{ftyp} > 0$ and $k_{f\nu nom} > k_{fmax}$. Note that efficient code will not execute the conditional in eq. (55a) once per data word. Instead, the accumulation calculations will be performed in one of three different iterative loops, depending on the value of Δk_{ftyp}. Additional economies can be had in the 1st and 3rd conditional clauses of eq. (55a). The value of $-\eta_{f\nu}$ for the 1st condition is equal to the sign bit of the 2's compliment representation of $k_{f\nu nom} - k_{fmin}$. Similarly, $+\eta_{f\nu}$ for the 3rd condition is equal to the sign bit of the 2's compliment representation of $k_{fmax} - k_{f\nu nom}$. In either case, $\eta_{f\nu}$ (or its equally useful negative) can be computed in 2 operations, and the conditional execution that is implied by the $min()$ and $max()$ functions in eq. (55a) is avoided.

Summary and Review of Index Calculations The following is a review and summary of the operations that compute the bit-wise parallel representations of the prompt and early-minus-late codes for an entire accumulation interval. The first operation is the iteration of eqs. (42a)–(42c) to construct the auxiliary table of $\Delta i(\mu)$ values. Next comes the computation of the auxiliary constants in eqs. (46a)–(46e) and (48a)–(48e). The third step is to initialize k_{f1} and μ_1 by evaluating eqs. (49a)–(50b). The last and costliest procedure is the iteration of eqs. (51a), (51b), (55a)–(55c), (47), (53), (43), and (54a)–(54c). Each iteration computes, successively, $k_{f\nu nom}$, $\mu_{\nu nom}$, $\eta_{f\nu}$, $k_{f\nu}$, μ_ν, k_ν, i_ν, $x_{p\nu}$, $x_{emlzm\nu}$, and $x_{eml2s\nu}$.

VI. 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 6 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. They use 6 *EXCLUSIVE OR* operations and 26 additional bit-wise logic operations per word.

They use 16 bit-summation operations and 16 additions per summation word. Suppose that the nominal word length is 32 bits but that the summation words are only 16 bits long. Then there are respectively 382 words and 763 summation words in a typical 1-msec L1 C/A code accumulation interval and twice these numbers for a typical 2-msec L2 CM/CL code accumulation interval. If one totals the necessary operations, then this method requires at least 36,640 operations per C/A code accumulation interval. For the CM/CL accumulation interval this method requires $\geq$73,216 operations.

Generating the over-sampled PRN codes requires additional computations. Generation of the C/A code requires 23 operations per data word, which assumes that the C/A code chips are pre-computed and stored in a table as in eq. (43). Generating the CM/CL code requires 36.7 operations per data word for the computations of Section V plus 29.5 operations per data word to generate the +CM/CL and –CM/CL codes using feedback shift registers as in [10]. Using the same number of words per accumulation as above gives a total of 8786 operations per 1-msec C/A accumulation interval and 50,511 operations per 2-msec CM/CL code accumulation. Thus, the real-time bit-wise parallel PRN code over-sampling imposes about a 46% increase in the computational load of the software receiver.

Bit-wise parallel correlation saves computation time in comparison to integer mathematical correlation operations. Integer mathematics requires the following calculations per channel per sample to generate the L1 and L2 in-phase and quadrature prompt and early-minus-late accumulations: 2 code time calculations, 2 prompt chip number determinations, 2 early chip number determinations, 2 late chip number determinations, 3 subtractions of late from early, 2 carrier phase evaluations, 4 carrier sine and cosine evaluations, 4 base-band mixing computations, 12 code mixing computations, and 12 accumulation summations. The total number of operations per sample is 45. Using the receiver sampling frequency of 12.199 MHz, the total number of operations per channel per 1-msec accumulation interval using integer mathematics is approximately 550,000. Totaling the number of operations from above for the bit-wise parallel software correlator gives approximately 110,000 operations per channel per 1-msec accumulation interval. This shows that the bit-wise parallel software correlation method is about 5 times more efficient than the brute-force integer method.

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. An increase above 1 bit for the RF data will make the logic more complex. For example, if the RF front end uses 2-bit digitization rather than 1-bit digitization, then the operation count for the mixing and accumulation operations will increase by a factor of almost 2, but the cost of the code over-sampling will remain constant. A 2-bit RF front end output data implementation of a GPS L1 software receiver is described in [3], [4].

A measure of the efficiency of the algorithms used in the software correlator is the number of accumulations per clock cycle. On an Intel Pentium 4 processor, each instruction requires one clock cycle (if data is loaded from memory). The L1 correlator processes 88 $\times 10^{-6}$ 1-ms accumulations per clock cycle, while the L2 correlator processes 65 $\times 10^{-6}$ 2-ms accumulations per cycle. The number of accumulations per clock cycle for the L1 correlator is less than twice that of the L2 correlator. Thus, the L2 correlator is slightly more efficient. This is the result of its re-use of indexing computations in its code over-sampling calculations.

Additional optimization of the correlation algorithms is possible. Using an Intel C compiler for Linux, off-line computational performance testing indicates that a significant speed-up on the Pentium 4 microprocessor is possible. Compared to the code produced by the GCC 3.3 compiler, the Intel compiler produced code that runs approximately 27% faster. This improvement in speed is most likely due to the Pentium-4-optimized code produced by the Intel compiler. The implementation of the software receiver reported on in this paper uses the GCC 3.3 compiler.

Two advances in microprocessors allow for increases in computational performance. First, as faster PC's become available computational performance will increase. Second, the advent of 64-bit microprocessors allows for twice as many correlation accumulations per clock cycle. Off-line performance testing of the software receiver on a 64-bit AMD Athlon 64 FX microprocessor running at 2.2 GHz indicates a computational performance gain of 28% over a 32-bit 3.2 GHz Intel Pentium 4 microprocessor. By factoring in the relative clock speeds of these two microprocessors it is clear that the 64-bit microprocessor is nearly 2 times faster than the 32-bit microprocessor in this application.

Another computational performance increase can be gained by using an RF front end that outputs two data streams at one-half the sampling rate instead of one data stream at the given sampling rate. In such an RF front end, the L1 C/A code signal and the L2 CM/CL code signal are separated into two data streams. Lower sampling rates are possible in such a system because each signal stream has half the information bandwidth. This reduces the computational requirements by a fac-

tor of 2, because the operation count is linearly proportional to the sampling rate.

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 382 32-bit words for the L1 C/A code signal and 763 32-bit words for the L2 CM/CL code signal. These sizes guarantee covering the full 12,199 and 24,398 RF front end samples for, respectively, the 1 ms C/A code accumulation and 2 ms CM/CL code accumulation intervals. Thus, 382×4=1528 bytes are required for each bit of each carrier signal that must be stored for the C/A code signal, and 763×4=3052 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 6112 bytes for the L1 carrier replicas and 12,208 bytes for the L2 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 L2 CM/CL signal, 179 Doppler shifts must be stored to cover the –8 KHz to +8 KHz range with a 87.5 Hz grid spacing. This translates into 2820 Kbytes of storage for all of the carrier replica signals.

The PRN code look-up tables require a modest amount of memory. The required memory for the tables in bytes is:

$$\begin{aligned} \text{Memory} = \quad & 3 \times ceil\left(\tfrac{n_s}{8}\right) \times 2^L \times k_{tot} \\ \approx \quad & 3 \times ceil\left(\tfrac{n_s}{8}\right) \times 4 \\ & \times 2^{[(n_s-1)\Delta t_s + \Delta t_{eml}]/\Delta t_{cnom}} \times \left(\frac{m\Delta t_{cnom}}{\Delta t_s}\right) \end{aligned} \tag{56}$$

The required amount of memory increases linearly with m and exponentially with n_s. Recall that n_s is the number of samples in a data word and that m is inversely proportional to the tabulated code start/stop time resolution $\Delta t_s/m$.

The memory requirements for the three PRN code tables have been calculated. These calculations assume that $\Delta t_{cnom} = (1/1.023 \times 10^6)$ sec, $\Delta t_{eml} = \Delta t_{cnom}/2$, and $n_s = 32$ samples per data word. This software receiver uses the following parameters: $\Delta t_s = 81.97$ nsec and $m = 14$. The range-equivalent code resolution is $c\Delta t_s/m = 1.76$ m. These values yield $L = 5$ code chips per data word, $k_{tot} = 170$ tabulated code start/stop times, and $2^L \times k_{tot} = 5440$ entries per PRN code table. The three PRN code tables together require 64 Kbytes 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 is relatively short. It is possible to use pre-computed tables of the C/A 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. Each pre-computed code is stored as a pre-computed auxiliary table index component table $\Delta i(\mu)$. Each auxiliary table has only 1029 1-byte elements. The total additional memory required to store the auxiliary tables for all 32 satellites is only 32 Kbytes.

Additional temporary memory is required to store the CM/CL code's auxiliary table. For $L \leq 8$ chips per data word the required memory equals $(M + 1 + L)$ bytes. For this receiver, $M = 2046$ and the auxiliary table requires 2 Kbytes.

The micro-computer stores the most recent 43 msec of RF front end data in a circular buffer. This buffer occupies 64 Kbytes of memory.

The total amount of memory required for storing the tables and data is 2982 Kbytes. The machine code for the software receiver requires additional storage on the order of 1500 Kbytes.

VII. RECEIVER PERFORMANCE RESULTS

Two scenarios have been used to test the software receiver's tracking and navigation performance. The first scenario is a non-real-time test in which the receiver operates on RF front-end output data that has been generated by a MATLAB simulation.

The tracking loop performance of the software receiver code has been evaluated by comparing its results to the simulated truth results. The software receiver's L1 and L2 code and carrier tracking loops run independently of one another. Fig. 8 compares the Doppler shift of the L1 and L2 carriers from the software receiver with that of the simulated data. The simulated data has high-bandwidth Doppler shift variations. It is clear from Fig. 8 that the two carrier tracking loops performed well.

The code-tracking performance of the software receiver can be determined by comparing the code start/stop times of the software receiver to that of the simulation. The difference of the start/stop times is the residual code phase error in seconds. Fig. 9 shows the residual code phase error for the L1 C/A and L2 CM signals. The errors have post-transient standard deviations of

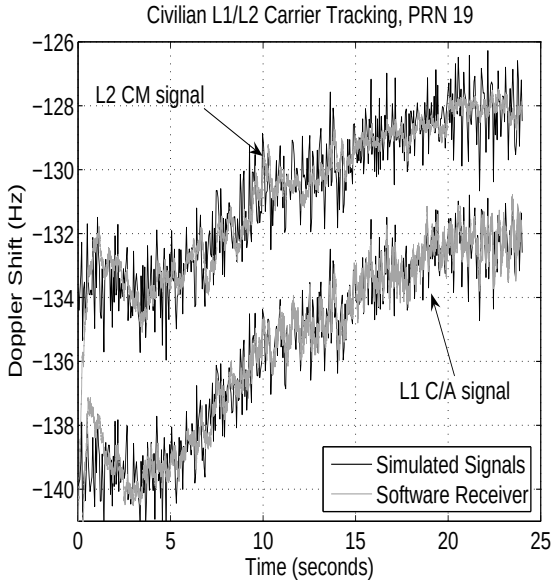

Fig. 8. Carrier Doppler frequency output of two independent frequency-locked loops tracking simulated L1 and L2 carrier signals. The L2 signal has been artificially biased to be closer to the L1 Doppler shift for plotting reasons.

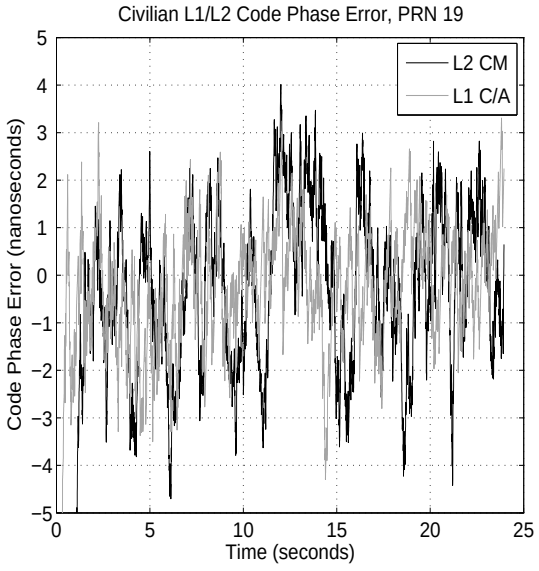

Fig. 9. Residual code phase error of two delay-locked loops tracking simulated L1 C/A and L2 CM signals.

approximately 1.3 ns (0.4 m). These results indicate that the software receiver is accurately computing correlation accumulations and accurately tracking the GPS signals.

The second test scenario is a real-time test. Testing has been performed using hardware-simulator-generated data. A simulator is required because the current GPS constellation does not include any GPS satellites that broadcast the new civilian L2 signal. Although GPS civilian L1/L2 simulators are available, Cornell University does not own one. Fortunately, Spirent Communications provided an opportunity to collect RF output data during a simulator demonstration. They demonstrated a GS7700 simulator, which is capable of generating signals containing the L1 C/A code and the L2 CM/CL code. The RF data was collected using a direct RF sampling front end configured in the same way as the one for the software receiver. The front end's 1-bit output RF data stream was bit-packed using shift registers, read into a PC using a DAQ card, and stored on a hard drive.

To simulate real-time data output from an RF front end, the stored data has been played back at the equivalent real-time rate using a digital data playback system running on a PC. The playback system has been configured in the following way. A PC has been set up with a digital output DAQ card installed and the recorded RF output data from the Spirent GS7700 simulator on its hard drive. The DAQ card is a National Instruments PCI-DIO-32HS DAQ card. This card has been configured to generate a clock signal at a rate of 12.199/32 MHz = 381.22 KHz. The outputs of the card are a clock signal and 32 data lines. On each rising edge of the clock signal, the card outputs 32 bits of data. This card is connected via a shielded cable to the DAQ card in the software receiver's PC. This DAQ card reads the 32 bits of data on each clock signal's falling edge. To the software receiver this input data looks exactly as it would if it had been acquired in real time using an RF front end.

The playback system has been used to test the software receiver's L1 and L2 acquisition and tracking capabilities and its navigation accuracy. The computed navigation solution for a dual-frequency GPS receiver should be accurate to within a few meters. The software receiver has been tested in two real-time configurations. In the first configuration, pseudorange measurements on the L1 signal have been corrected using the ionospheric model that is transmitted with the navigation message. In the second test configuration, the L1 signal's pseudorange measurements have been corrected using the measured L1/L2 differential code delay.

Fig. 10 shows the navigation solution residual error with respect to the simulator's truth position for both test configurations. The navigation solution residual error for the two test configurations are roughly equivalent. This level of residual error is what one would expect from a dual-frequency receiver with code-phase-based navigation. The L1-only navigation solution with

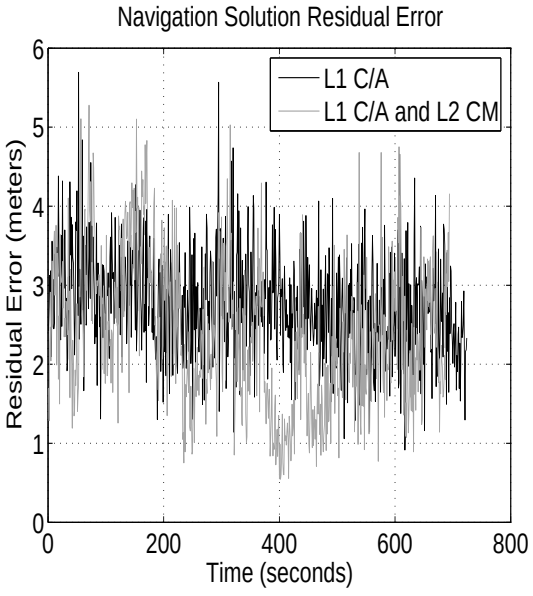

Fig. 10. Navigation solution residual error with respect to the truth position output by the simulator.

modeled ionospheric corrections is almost as accurate as the dual-frequency-corrected navigation solution because of how the Spirent simulator generated its ionospheric group delay. It was configured to generate ionospheric group delays using the ionospheric model that is used by the broadcast corrections. The same model parameters were output in the navigation data stream as were used in the simulator's model. This allowed the L1-only test configuration to successfully correct the ionospheric group delay, resulting in a more accurate navigation solution than nominally would be available from a single-frequency receiver.

The ad hoc dual-frequency correction algorithm implemented in this receiver is not optimal. A least-squares or Kalman-filter-based technique may generate more accurate estimates of the ionospheric group delay. The present method has been used because of its ease of implementation and because of development time constraints.

VIII. FUTURE WORK

Four significant improvements remain to be made to the current receiver. First, a more intelligent and faster signal acquisition method is needed for the CM/CL code. This may be either an FFT-based method or a traditional time-domain search method that makes better use of L1 C/A code tracking information.

Second, carrier- and code-tracking loops for the L2 CL code need to be developed and integrated into the receiver. Since the CL code does not contain navigation data, it is useful for weak-signal acquisition and tracking.

Third, decoding has not been implemented for the CM code navigation data stream. It would be advantageous to add this feature to the software receiver. The primary benefit of this data stream is that error detection and correction capabilities are more robust than those present on the C/A code's navigation data stream.

Fourth, the software receiver's computational performance could be improved. Significant improvement would offer an increase in the number of tracking channels. One plan is to migrate the software to a 64-bit computing platform.

Additional performance gains are possible by using dual microprocessors or dual-core microprocessors. To take advantage of these microprocessor architectures, the software correlator would need to be modified to run as a multi-threaded application. This modification is straightforward and could easily be implemented.

IX. SUMMARY AND CONCLUDING REMARKS

A 10-channel real-time GPS civilian L1/L2 software receiver that runs on a PC has been implemented and tested. The hardware consists of a direct RF sampling front end, a data buffering and acquisition system, and a PC with a 3.2 GHz Intel Pentium 4 processor running RTAI. The software consists of a dual-frequency real-time software correlator and GPS software that provides the typical GPS functions such as signal acquisition, signal tracking, and navigation. The software correlator, running on the PC's processor, consumes about 80% of the CPU capacity. The navigation accuracy of this receiver is on the order of 2–5 m.

ACKNOWLEDGEMENTS

The authors would like to thank Spirent Communications for providing data to test this software receiver. This research was supported in part by ONR grant number N00014-92-J-1822.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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*, To Appear, 2004.

[5] M. L. Psiaki, "Design and Practical Implementation of Multi-Frequency RF Front Ends Using Direct RF Sampling," *Proc. of the Institute of Navigation GPS*, Portland, OR, Sept. 9–12, 2003, pp. 90–102.

[6] 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.

[7] J. A. Klobuchar, "Ionospheric Effects on GPS," in *Global Positioning System: Theory and Applications, Vol. I*, B. W. Parkinson and J. J. Spilker Jr. , Eds., American Institude of Aeronautics and Astronautics, (Washington, 1996), pp. 485–515.

[8] R. D. Fontana, Cheung, W., Novak, P.M., and Stansell, T.A., "The New L2 Civil Signal," *Proceedings of the Institute of Navigation GPS*, Sept. 11–14, 2001, Salt Lake City, UT, pp. 617–631.

[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 Institude of Aeronautics and Astronautics, (Washington, 1996), pp. 329–407.

[10] Anon.,"Navstar GPS Space Segment/Navigation User Interfaces," ICD-GPS-200, Revision C, ARINC RESEARCH CORPORATION, El Segundo, CA, Jan. 2003.