GNSS Receiver Implementation on a DSP: Status, Challenges, and Prospects

Todd E. Humphreys, Mark L. Psiaki, and Paul M. Kintner, Jr., *Cornell University, Ithaca, NY*
Brent M. Ledvina, *Applied Research Laboratories, Austin, TX*

BIOGRAPHIES

Todd E. Humphreys is a graduate student in the Sibley School of Mechanical and Aerospace Engineering. He received a 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.

Mark L. Psiaki is a Professor in the Sibley School of Mechanical and Aerospace Engineering. 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.

Paul M. Kintner, Jr. is a Professor of Electrical and Computer Engineering. 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.

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 GNSS technology and applications.

ABSTRACT

A real-time GPS L1 C/A-code software receiver has been implemented on a Digital Signal Processor (DSP). The receiver exploits FFT-based techniques to perform autonomous acquisition down to a threshold of $C/N_0 = 33$ dB-Hz. Efficient correlation algorithms and robust tracking loops enable the receiver to track an equivalent of 43 L1 C/A-code channels in real time with a tracking threshold of 25 dB-Hz. This accomplishment represents a milestone in an ongoing effort to develop a low-cost, flexible, and capable GNSS receiver for use as a scientific instrument and for GNSS receiver technology development. This paper reports on the current design and capability of the DSP-based receiver, provides an overview of the challenges that are particular to embedded GNSS software receiver design, and discusses the prospects of DSP-based GNSS software receivers in relation to the multiple frequencies and higher bandwidths offered by modernized GNSS.

INTRODUCTION

Over the last five years, several researchers have capitalized on the software radio concept to design GPS receivers whose real-time correlators, tracking loops, and navigation solver are all implemented in software on a programmable processor.[1–6] Software GNSS receivers require fewer hardware components and offer greater flexibility compared to traditional receivers whose correlators and accumulators are implemented in dedicated integrated circuit hardware.

Originally used only for post-processing of GPS signals, software receivers broke into the real-time domain with the introduction of the gpsSrx receiver, first implemented on a personal computer (PC) microprocessor and then on a 160-MHz DSP.[1,2] The latter was the first implementation of an *embedded* real-time GPS software receiver, where the term embedded is used to distinguish small, low-power, stand-alone receivers—implemented, for example, on a DSP, an FPGA, or a small applications processor—from software receivers implemented on a PC.

The current work can be viewed as modernization and an extension of the work reported in Ref. 1. Whereas the receiver in Ref. 1 was limited to 4 real-time L1 channels, separate acquisition and tracking stages, and modest (30 m) positioning accuracy, the current receiver is capable of 43 parallel L1 channels, continuous background acquisition, and 5-m positioning accuracy (position solutions are calculated off-line using the RINEX-type files produced by the receiver). The receiver owes these advances to increased DSP capability and to improved processing techniques.

Embedded software GNSS receivers are attractive to the GNSS research community both as a platform for receiver technology development and as a scientific instrument useful for studying the ionosphere and neutral atmosphere. The arrival of a second civilian signal at L2—broadcast presently by only two Block IIR-M GPS satellites but projected to be available from 6-8 satellites by about December of 2007[7]—offers the prospect of a low-cost, dual-frequency, science-grade GNSS receiver. Such an instrument will be an essential component in future work that calls for large arrays of GNSS receivers whose combined measurements will be used to investigate the spatial irregularity, dynamics, and height of structures in the disturbed ionosphere.

This work will focus on the DSP as the target for an embedded GNSS receiver implementation. Alternatives to the DSP include general applications processors (e.g., Intel's XScale and TI's OMAP processors) and FPGA and FPGA/DSP hybrid platforms.[5,8] The authors favor the DSP over applications processors because high-performance DSPs are currently better suited for carrying out the demanding correlation operations required in a GNSS receiver (although with the introduction of the OMAP III the DSP's advantage is thinning). Preference for the DSP over the FPGA and FPGA/DSP hybrids is motivated by simplicity: the DSP is easier to program than the FPGA, and the additional throughput that the FPGA offers is unnecessary for L1 C/A-code receivers and likely unnecessary for GPS L1 + GPS L2C + Galileo L1 BOC(1,1) receivers. On the other hand, processing the wideband civil GNSS signals at L5/E5 may only be practical in the short term on an FPGA or FPGA/DSP hybrid platform.

The remainder of this paper is divided into eight sections. These are listed here for ease of navigation:
I: Embedded Platform Considerations
II: Correlation, Acquisition, Tracking, and Calculation of Observables
III: Code Architecture
IV: Further Details on Implementation
V: Code Development and Testing Methodology
VI: Performance
VII: Prospects
VIII: Summary

I. EMBEDDED PLATFORM CONSIDERATIONS

A. Overview of the DSP-based GNSS receiver Platform

The DSP-based GNSS receiver platform on which this work reports is shown schematically in Fig. 1; a photograph of the prototype hardware is seen in Fig. 2. The RF front end consists of a Zarlink GP2015 and supporting hardware. The GP2015 mixes the incoming L1 signal to 4.309 MHz and then samples at 5.714 MHz, yielding a 2-bit sampled IF centered at 1.405 MHz. Separate sign and magnitude data streams are fed directly into the Texas Instruments (TI) TMS320C6416 DSP ('C6416) via two independent multi-channel buffered serial ports (McBSPs). A series of binary counters and AND gates (seen on the prototyping board in Fig. 2) is used to generate a synchronizing pulse for the McBSPs every 32 clock cycles. A DSP Starter Kit (DSK) board has been used for ease of development. The components of the DSK board that are essential to the receiver are the DSP chip (outlined in red) and the off-chip SDRAM (located just above the DSP). Subsequent sections will comment on the features of DSPs in general and on those of the 'C6416 specifically.

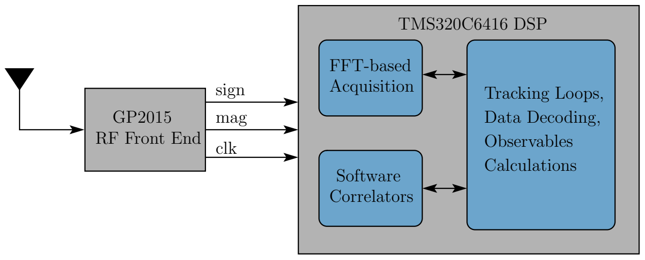


Fig. 1. Schematic representation of the DSP-based GNSS software receiver.

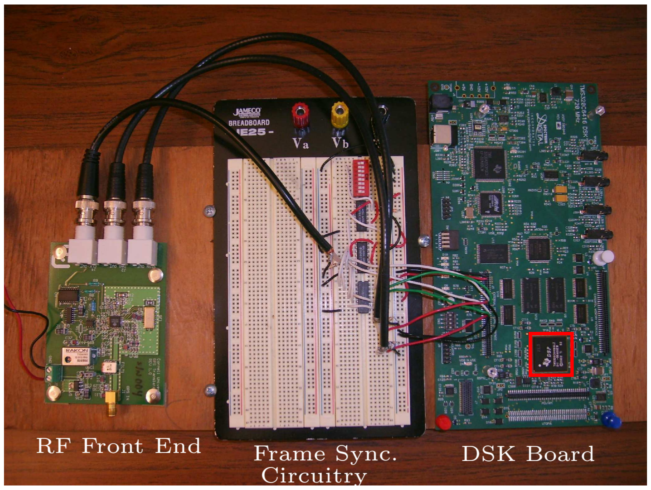


Fig. 2. Photograph of the DSP-based GNSS software receiver prototype. The DSP chip is outlined in red on the DSP Starter Kit (DSK) board.

B. Digital Signal Processors (DSPs)

The DSP can be thought of as a specialized processor designed to support repetitive, numerically intensive tasks.[9] As such, the DSP may be contrasted with a general-purpose processor, such as is found in a modern desktop PC, which is designed to handle a variety of hardware management tasks in addition to the processing of data. In truth, however, DSPs are no longer so clearly differentiated on the basis of processor architectures. This is because the increased importance of signal processing has motivated general-purpose processor vendors to add DSP functionality to their processors (e.g., the single-instruction, multiple-data instruction set extensions in the Pentium MMX processors).[10] Nonetheless, the highest performance DSPs still retain a decided advantage in power consumption over x86-type processors ($\sim$1 W vs. $\sim$20 W), and an advantage in processing capability over general-purpose embedded applications processors (e.g., ARM, XScale, OMAP).

Several features of the DSP make it well suited as a GNSS software receiver processor. First, the DSP's signature feature—the single-instruction multiply-and-accumulate (MAC)—speeds up the correlation operation, which constitutes the bulk of the processing demand in a real-time GNSS software receiver. Second, because of the widespread use of the Fast Fourier Transform (FFT) in signal processing, the DSP's addressing modes have been designed to enable efficient FFT execution. This feature can be used to speed acquisition in a software GNSS receiver. Indeed, in the original paper that introduced FFT-based acquisition to the GPS community, a DSP was used to implement the acquisition routine.[11] Third, the modern DSP's extensive suite of peripherals (e.g., timers, external memory interfaces, direct memory access, serial ports, etc.) offloads the I/O and some housekeeping tasks from the DSP's CPU and makes interfacing with the DSP straightforward.

Of course, the DSP also has its limitations as a processor for a GNSS software receiver. Most notable are the DSP's limited on-chip memory and the fact that the DSP's vaunted high throughput can only be realized using fixed point arithmetic. Section IV treats these challenges in more detail.

C. The TMS320C6416

The decision to adopt the Texas Instruments TMS320C6416 DSP as the target for the DSP-based GNSS receiver was made after an earlier attempt failed to develop a receiver on a smaller, less expensive, but less capable target—the Analog Devices Blackfin DSP. Thereafter, the authors reformulated the development question: instead of asking whether a particular software GNSS receiver could be implemented on a particular DSP, they expanded the question to ask, "What can be done on the *best available* DSP?" When this phase of work began, the best available DSP was arguably the 720 MHz TMS320C6416.[12]

The 'C6416 is a fixed-point, very long instruction word (VLIW) DSP with 1 MB on-chip memory that is targeted for high-performance applications.[13] It has two independent data paths, each with four execution units. The eight execution units are capable of executing up to eight 32-bit instructions in parallel. For example, the 'C6416 can perform eight 8-bit multiplications in parallel. Clearly, such an architecture is well suited for the demanding correlation operations involved in a GNSS software receiver.

Besides the DSP chip itself, the 'C6416 is well supported by TI's integrated development environment and by adequate documentation. The optimizing compiler accepts ISO standard C source code and, with minor limitations, ISO standard C++ source code.[14] This allows code to be written and tested on a PC and then quickly ported to the DSP. It also means that the code can be easily transferred to other DSPs that support C/C++. Section VII considers alternatives to the 'C6416.

D. Tradeoffs in Embedded Software Design

As is common in embedded software design, the challenge of squeezing a GNSS software receiver into a DSP is primarily one of negotiating pairwise or groupwise tradeoffs between desirable but conflicting performance measures. One would like to design embedded code that is at once memory-efficient, flexible, readable/maintainable, and, above all, fast. Unfortunately, an improvement in one of these characteristics tends to diminish one or more of the others. Storing local code and carrier replicas in lookup tables is, for example, much faster than generating them in real time, but requires more memory. Similarly, implementing assembly code as opposed to C/C++ enhances speed and memory use at the expense of flexibility and readability/maintainability.

The best approach to negotiating these tradeoffs is not necessarily to seek out some delicate balance between the opposing performance measures, but rather to seek out devices and algorithms that allow one to cheat the tradeoffs by improving one aspect of performance without a significant detriment to the others. FFT-based acquisition and bit-wise parallel correlation, introduced in the next section, are two examples of such algorithms.

II. CORRELATION, ACQUISITION, TRACKING, AND CALCULATION OF OBSERVABLES

A. Bit-wise Parallel Correlations

The bit-wise parallel correlation methods introduced in Ref. 15 were implemented on the DSP-based GNSS receiver and proved to be a key to the performance results reported in this paper. The following paragraphs describe the adaptations made to the bit-wise approach of Ref. 15 for implementation on the 'C6416.

In the DSP-based receiver, the bit-wise correlation method is based on coherent accumulations spanning one C/A code interval (1 ms). Longer coherent accumulations are synthesized by adding successive 1-ms accumulations after performing a "fix-up" rotation of the (I, Q) vector so that resulting sum is phase coherent. At a sampling rate of $F_s = 5.7143$ MHz each 1-ms accumulation nominally involves $N_s = 5714.3$ data samples from the RF front end. In the bit-wise parallel correlation approach, the N_s sign and magnitude bits are packed into 32-bit words and correlation is performed in parallel on 32 bits. In the DSP implementation, the N_s samples are truncated to

$32 * \text{floor}(N_s/32)$ samples to optimize the efficiency of the pipelined correlation loop. For $N_s = 5714.3$, this amounts to a negligible 0.03-dB loss in signal power. Higher sampling rates (as required for processing the Galileo L1 BOC(1,1) signal) further lessen the effect.

In the current DSP implementation, local carrier replicas are precomputed and stored in on-chip memory. This approach sets up a tradeoff between frequency resolution and memory use. To examine this tradeoff, let Δf be the maximum frequency difference between the incoming Doppler-shifted carrier and the nearest local carrier replica, and let $\theta_n = \pi \Delta f T$ be half the maximum rotation angle of the (I, Q) vector over a $T = 1$ ms accumulation. Figure 3 shows the loss of signal power as function of θ_n. For tracking, it was decided to limit power loss to 0.11 dB, which corresponding to a loss factor of 0.975 (indicated by the upper horizontal line in Fig. 3). This implies a maximum Δf of 87.5 Hz, or a frequency spacing of $2\Delta f = 175$ Hz. Using a simple compression scheme, local carrier replicas at this frequency spacing and spanning a ±10 kHz occupy a modest 81 kB of on-chip memory (cf. Fig. 9). For acquisition, the search frequency spacing is doubled to 350 Hz, which corresponds to a maximum 0.442-dB loss in signal power and a loss factor of 0.9032 (indicated by the lower horizontal line in Fig. 3).

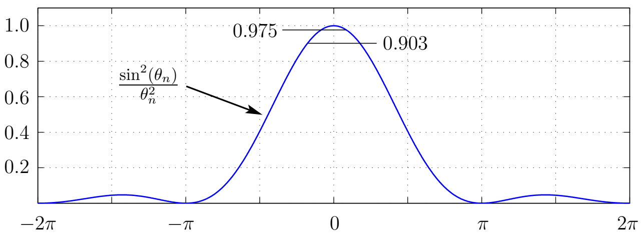


Fig. 3. Normalized power as a function of the half rotation angle θ_n.

The next adaptation of the bit-wise method presented in Ref. 15 deals with the quantization of the local carrier replicas. The quantization values used in Ref. 15, which are $\{-2, -1, 1, 2\}$, reflect the original levels used in the GP2021 chip for which the bit-wise correlation method is a software substitute. But these values are sub-optimal for quantization of the sinusoidal carrier replicas. As shown in Fig. 4, the optimal 2-bit quantization scheme for sinusoids involves a selection of three parameters, a_0, a_1 and L, leading to a 3-parameter optimization of the form

$$J(a_0, a_1, L) = \int_0^{2\pi} \{q[\sin(\theta); a_0, a_1, L] - \sin(\theta)\}^2 \, d\theta$$

where $q[\sin(\theta); a_0, a_1, L]$ is the quantized value of $\sin(\theta)$ with quantization parameters a_0, a_1 and L. The cost function J is minimized subject to the constraints $0 \leq a_0 \leq L \leq a_1$ and

$$\frac{a_1}{a_0} = \frac{k_1}{k_0}$$

where k_1 and k_0 are small integers. The latter constraint is applied for practical implementation on the DSP. It ensures that the products of the quantized carrier replicas and the quantized front-end data can be stored in signed 8-bit elements

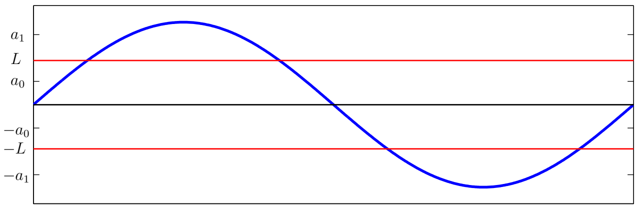


Fig. 4. Quantization values $\{-a_1, -a_0, a_0, a_1\}$ and thresholds $\{-L, 0, L\}$ for quantization of local carrier replicas.

Using numerical techniques, the quantization parameters that minimize J subject to the constraints are easily found: $a_0 = 0.283, a_1 = 0.849$, and $L = 0.536$, with $k_0 = 1$ and $k_1 = 3$. As it turns out, these are the same k_0 and k_1 used by the GP2015 RF front end, which means that the current implementation of bit-wise parallel correlation saves two 32-bit multiply-and-accumulate operations per iteration of the correlation loop compared to Ref. 15 because there are two fewer combinations of incoming data amplitudes and carrier replica amplitudes.

The final adaptation of the bit-wise technique to the DSP exploits the `bitc4()` instruction, one of the 'C6416's specialized signal processing instructions. The `bitc4()` instruction is used to accumulate the number of 1s bits in a 32-bit word, thus obviating the 64-kB look-up table used for the same purpose in Ref. 15.

B. FFT-based Acquisition

The sequential search technique for GNSS signal acquisition is the simplest to implement and requires very little memory beyond what is used for signal tracking, but it is computationally expensive. This is especially true if one wishes to increase acquisition sensitivity by extending the coherent or noncoherent integration time. Moreover, sequential search methods become impractical for the higher sampling rates and longer spreading codes of the Galileo L1 BOC(1,1) signal. Fortunately, the DSP is well suited to FFT-based acquisition.

Like other DSP manufacturers, Texas Instruments provides a library of FFT routines that have been optimized for use on the DSP.[16] For computational efficiency, these routines are uninterruptible, they require the number of points N in the FFT to be a power of 2, and they require N 16-bit complex factors to be computed beforehand

and stored in (preferably on-chip) memory. The following FFT-based acquisition strategy has been designed around these restrictive but efficient FFT library functions.

B.1 Cost/Sensitivity Analysis

Acquisition sensitivity is increased by increasing the coherent or noncoherent averaging time in each search cell. Increasing the coherent averaging time T_c is more effective than increasing the number K of noncoherent averages (as measured by the length of the data interval required to achieve a given acquisition threshold), but, for several reasons, increasing T_c beyond a certain limit becomes computationally impractical.

Consider the following cost/sensitivity analysis for FFT-based acquisition on the 'C6416 using the DSP Library function `fft16x32()`. It can be shown that the total computation time required for acquisition on one channel with $F_s = 5.714$ MHz, $T_c = 1$ ms, $N = 8192$, N_f Doppler search frequencies, and K noncoherent integrations (with all data stored on chip) is given by

$$T_{acq} = 182 + 310 + 415N_f + KN_f(204 + 310 + 310)$$

where time is expressed in μs. In the above equation, the quantities 182, 415, and 204 scale linearly with N, whereas the three 310s, which represent FFT calculations, scale as[16]

$$T_F = C_t \left[\left(\frac{13N}{8} + 24 \right) \text{ceil} \left[\log_4(N) - 1 \right] + \frac{3(N+8)}{2} + 27 \right]$$

with $C_t = 3.36 \times 10^{-3}$. As T_c increases, N increases to the minimum power of 2 that can accommodate $N_s = TF_s$ front-end samples (set aside for now issues of limited memory and limitations on the maximum value of N). Also, as T_c increases, N_f must increase by the same factor to avoid loss of signal power over the coherent integration interval. Finally, as T_c increases, the probability of encountering a navigation data bit transition increases. To approximate this event as an equivalent computational penalty, assume that each coherent integration interval that straddles a navigation bit start/stop time must be discarded and replaced by another coherent integration interval of equal length. Thus for $T_c = 1$ ms and $K = 20$, a total of 21 1-ms coherent intervals must be computed, and for $T_c = 10$ms and $K = 2$, a total of 3 10-ms coherent intervals must be computed.

To complete the cost/sensitivity analysis, assume that the number K of noncoherent integrations is chosen at each carrier-to-noise (C/N_0) ratio to be the minimum K that guarantees "reliable" acquisition, where "reliable" is interpreted as a probability of detection $P_D \geq 0.95$ and a probability of false alarm $P_{FA} = 0.01$. (Note that P_{FA} is the probability that *any* cell in the $N_f \times N$ search space exceeds the acquisition threshold. For an introduction to the hypothesis testing methods used to calculate P_{FA} and P_D, see Ref. 17, Appendix B. Note that there is a sign error in the argument of the exponential function in Ref. 17's Eq. (B7): the term $-2K\beta$ should be $+2K\beta$.)

The above analysis can be combined with an assumed Doppler resolution of $N_f/T_c = 35$ bins per ms to generate the plot shown in Fig. 5, which gives the computation time as a functin of C/N_0 for several different values of T_c. (Note that computation time as expressed in Fig. 5 should not be confused with the length of the front-end data interval required for acquisition: the latter is much less than the former.) Figure 5 indicates that an acquisition strategy based on $T_c = 1$ ms is computationally more efficient than strategies based on longer coherent integration times for values of C/N_0 down to 24 dB-Hz. Adding to this result the facts that a longer T_c requires more memory and that the DSP Library function `fft16x32()` limits N to 2^{15}, the choice of acquisition strategy becomes clear: GPS L1 C/A-code acquisition in the DSP should be based on noncoherent sums of $T_c = 1$-ms coherent integration intervals. Strategies for acquisition of L2C and Galileo signals will be discussed in Section VII.

Figure 5 is also useful for choosing a practical acquisition threshold. In the current implementation, a threshold of $C/N_0 = 32$ dB-Hz was chosen to limit the computation time for each PRN to less than 2 seconds.

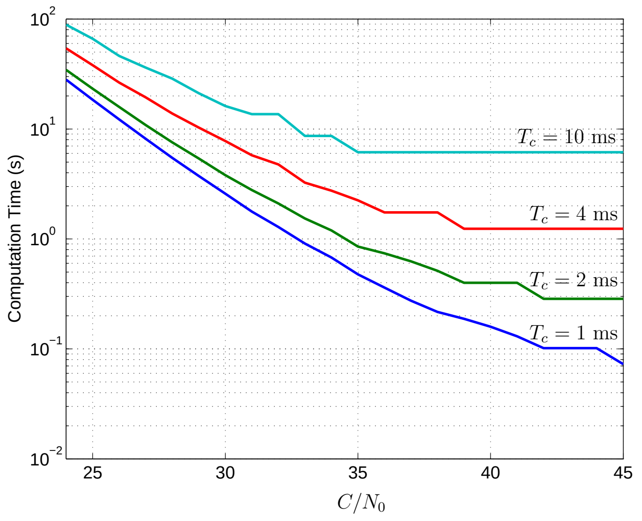


Fig. 5. Computation time required for reliable acquisition as a function of C/N_0 for several values of the coherent integration interval T_c.

B.2 Implementation

The implementation of FFT-based acquisition in the DSP is illustrated in block diagram form in Fig. 6. Before circular correlation, the front-end sign and magnitude bits must be converted to integers for use in the FFT and resampled using linear interpolation to arrive at an even power of 2.

These operations represent an inefficiency relative to the bit-wise correlation strategy, which operates directly on the 5714 sign and magnitude bits, but the DSP handles the conversion and resampling quickly (24 μs to convert 5174 front-end samples and 52 μs to interpolate these into 8192 samples), and the inefficiency is more than compensated by the efficiency of the FFT-based circular correlation. As an alternative to resampling, one might consider zero-padding, but this requires $N = 16384$ and increases memory use.

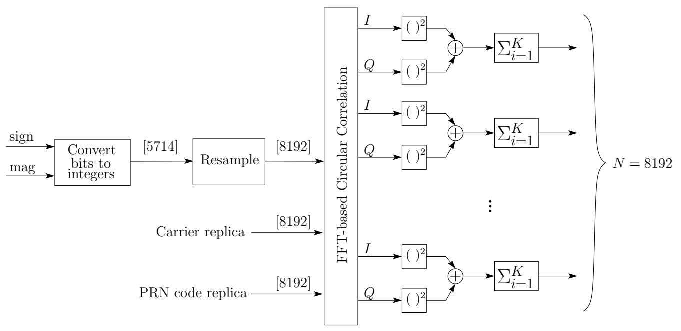


Fig. 6. Block diagram of the FFT-based acquisition implementation in the DSP

The results of the circular correlation operation are squared, summed, and accumulated noncoherently K times at each Doppler search frequency. If the maximum value of the resulting statistic across all frequencies and code offsets exceeds a predetermined threshold, then a successful acquisition is declared and the corresponding Doppler frequency and code offset are sent to the tracking loops. During real-time operation with a CPU laboring to track several channels, this "maximum value" approach can yield Doppler frequency and code phase estimates that are up to 2 seconds older than the incoming data. This latency does not obsolete the Doppler frequency estimate, which varies little over 2 seconds, but it does require the code phase estimate to be propagated forward to the epoch of the current incoming data. This propagation makes use of the following relation:

$$N_s = \frac{N_{s,0}}{1 + \frac{s_m f_D}{f_{L1}}}$$

where N_s is the value for the number of samples per C/A code period that is used for propagation, $N_{s,0}$ is the nominal number of samples per C/A code period assuming zero Doppler shift and a perfect receiver clock, $s_m = -1$ (1) for high (low)-side mixing in the RF front end, f_D is the apparent Doppler shift as measured by acquisition, and $f_{L1} = 1575.42$ MHz is the L_1 carrier frequency. As an alternative to the "maximum value" approach, one might terminate the search process as soon an any cell exceeds the predetermined threshold, thus eliminating the need for propagation. This method, however, suffers from a higher incidence of false alarm because of minor correlation peaks in the code offset dimension and side lobes in the frequency dimension (cf. Fig. 3). Hence, the "maximum value" approach is preferred. For a maximum frequency error of $\Delta f = 175$ Hz, the above propagation relation is accurate to 0.23 C/A code chips over a propagation interval of 2 seconds.

The acquisition search strategy implemented in the DSP begins by performing a quick search for strong signals ($C/N_0 \geq 42$ dB-Hz) using $K = 2$. Thereafter, K is gradually increased to 41. In the steady-state, the search strategy allocates 65% of its time to deep searches ($K = 41$) and 35% of its time to shallow searches ($K = 2$). The total on-chip memory footprint for the C/A-code FFT-based acquisition is 240 kB (cf. Fig. 9).

C. Tracking Loops

In the design of the DSP-based receiver, much attention has been focused on the tracking loops. This is in keeping with the receiver's envisioned role as a science-grade receiver that is capable, for example, of providing accurate and robust phase and amplitude tracking of ionospheric scintillations. Figure 7 gives a schematic of the receiver's signal tracking strategy; details on each loop follow.

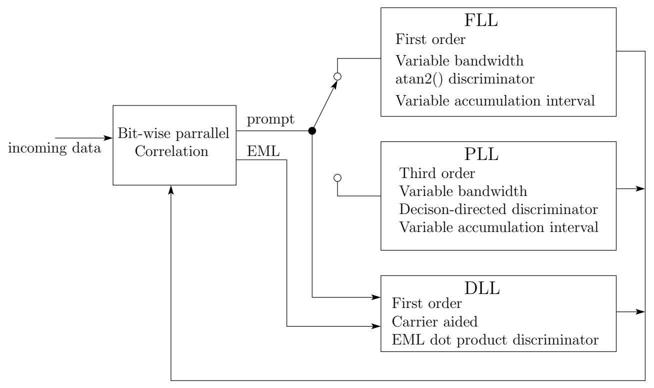

incoming data
Bit-wise parrallel Correlation
prompt
EML
FLL
First order
Variable bandwidth
atan2() discriminator
Variable accumulation interval
PLL
Third order
Variable bandwidth
Decison-directed discriminator
Variable accumulation interval
DLL
First order
Carrier aided
EML dot product discriminator

Fig. 7.

C.1 The Frequency-Locked Loop

The frequency-locked loop (FLL) is used as a bridge between signal acquisition and phase-locked loop (PLL) tracking. The first-order FLL employs a four-quadrant arctangent discriminator whose output δf is given by

$$\delta f = \text{atan2}(IQ_{cross}, IQ_{dot})$$

with $IQ_{cross} = I_k Q_{k+1} - I_{k+1} Q_k$ and $IQ_{dot} = I_k I_{k+1} + Q_k Q_{k+1}$, where I_k and Q_k are the prompt in-phase and quadrature accumulations at time t_k.

The FLL begins operation in either a nominal signal mode or a weak signal mode, depending on whether the initial C/N_0 estimate provided by the acquisition routine is above or below 33 dB-Hz. In the nominal signal mode, the FLL's closed-loop bandwidth is 5 Hz; in the weak signal mode it is 1 Hz. Prior to data bit synchronization, the FLL operates on 1-ms coherent accumulations, which, in combination with the four-quadrant arctangent discriminator, yields a pull-in range of ± 500 Hz. Such a wide pull-in range makes the FLL remarkably robust, enabling reliable bit synchronization down to $C/N_0 = 25$ dB-Hz.[17] Before bit synchronization, data bit transitions introduce noise into the FLL's Doppler frequency estimate because the four-quadrant arctangent discriminator is sensitive to 180-degree changes in phase. The noise is well within the FLL's pull-in range, however, and does not threaten the loop's stability.

After five FLL time constants have elapsed, the FLL attempts data bit synchronization using a standard histogram approach.[17] In nominal signal mode, the upper and lower histogram thresholds are $\tau_1 = 12$ and $\tau_0 = 7$, which, for $C/N_0 \geq 33$ dB-Hz, leads to a probability of detection near unity and negligible probabilities of bit sync error and false alarm. In weak signal mode, the upper and lower thresholds are $\tau_1 = 80$ and $\tau_0 = 70$, which, for $C/N_0 \geq 25$ dB-Hz, leads to a probability of detection of 0.64, a probability of bit sync error (signal present) of 0.005, and a probability of false alarm (no signal present) of 0.025.

If bit lock is successful, the FLL begins a graded transition from 1-ms accumulations to the nominal accumulation interval T_a. At each intermediate accumulation interval, five FLL time constants are allowed to pass. This approach ensures that the noise at low C/N_0 does not cause the FLL to break lock as its pull-in range narrows. Five FLL time constants after T_a is reached, frequency lock is declared and tracking control is transferred from the FLL to the PLL.

C.2 The Phase-Locked Loop

The DSP-based receiver's PLL is a variable-bandwidth third-order Costas loop with a decision-directed discriminator. The PLL's design is the outcome of a previous research effort in which the authors determined the loop structures and parameters that enable good phase tracking during ionospheric scintillations.[18] It was shown in Ref. 18 that a variable-bandwidth phase tracking loop based on a Kalman filter gave the best overall performance and was especially effective (relative to competing designs) at low C/N_0. However, because of the Kalman filter's prohibitive computational requirements, it was not implemented directly. Instead, a third-order Costas loop whose bandwidth can adapt to changes in C/N_0 was implemented. This approximation, along with the decision-directed phase discriminator, yields phase tracking performance that is very similar to that of the Kalman filter tracking loop.

C.3 The Delay-Locked Loop

The delay-locked loop (DLL) is a standard first-order carrier-aided 0.5-Hz-bandwidth code tracking loop with a dot-product discriminator whose output $\delta\tau$ before normalization is given by

$$\delta\tau = I_{E-L} I_P + Q_{E-L} Q_P$$

where I_{E-L}, Q_{E-L}, I_P, and Q_P are the early-minus-late and prompt in-phase and quadrature accumulations.

D. Calculation of Pseudorange and Carrier Phase

Pseudoranges for all channels are calculated simultaneously at the beginning of one channel's next C/A code period. Receiver time for purposes of pseudorange calculation is redefined at each pseudorange measurement epoch to be equal to the satellite transmit time for the first occupied channel. Hence, one pseudorange value is always equal to zero. For proper carrier phase measurements, the carrier phase in all channels is expressed as a phase offset from a common carrier phase reference. Carrier phase is measured simultaneously with pseudorange across all channels. A second receiver time expressed as the total number of samples and fractional samples that have been generated by the RF front end is output with the observables to provide a consistent time base for the carrier phase measurements. Figure 8 shows the outputs that the receiver sends to the user interface.

III. CODE ARCHITECTURE

Code for the DSP-based receiver is written in C++ and designed with an object-oriented approach. Historically, embedded code has been strictly limited to assembly language or C, but modern compilers are increasingly supporting C++, with minor limitations (cf. Ref. 14, Ch. 7). For portability, readability, and ease of maintenance, there was a strong incentive within the project to prefer a high-level source language such as C or C++ over assembly language. In choosing between C and C++, it was found that the TI compiler produces equally efficient object code from either source. Hence, there appeared to be no reason not to use C++ and a modern object-oriented approach in the design of the software receiver source code.

The code architecture for the DSP-based receiver is designed around the `Channel` class, which encapsulates all data specific to one GNSS signal (e.g., PRN, current code phase, carrier phase, Doppler frequency estimate, status, etc.) with member functions that perform acquisition, tracking updates, and data reporting. The `Channel` class objects themselves contain `FLL` (frequency-locked loop), `PLL` (phase-locked loop), and `Navdat` (for parsing and storing navigation data) class objects. Independent supporting functions combine data reported by each `Channel` object to compute pseudorange and carrier phase observables. Such an object-oriented implementation makes the code relatively easy to read and modify, and avoids the proliferation of stray global variables.

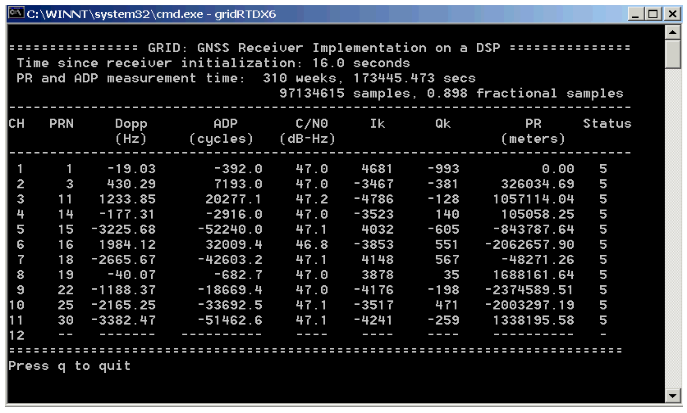
IV. FURTHER DETAILS ON IMPLEMENTATION

A. Input

The sign and magnitude data streams produced by the GP2015 RF front end are fed directly into two independent multi-channel buffered serial ports (McBSPs) on the DSP.[19] The McBSPs buffer the incoming data in a series of 32-bit shift registers, sending a synchronization event to the Enhanced Direct Access Memory (EDMA) coprocessor when the data receive register is full and can be read. On the ‘C6416, the EDMA supports up to 64 independent channels, only two of which are used in the current implementation: one for sign and one for magnitude RF front-end data. The EDMA fetches the 32-bit sign and magnitude words and stores these in independent circular buffers. When it reaches the end of a circular buffer, the EDMA sends a Hardware Interrupt (HWI) to the CPU and promptly renews operation at the beginning of the circular buffer. The HWI interrupt prompts the CPU to increment a loop counter used to keep track of the total number of front-end samples collected. Servicing the HWI is the only intervention required of the CPU in the data input process.

B. Output

Using TI’s Real-Time Data Exchange (RTDX) capability, information is sent from the DSP to a host computer while the receiver is in operation. An executable on the host computer displays the data to the screen (Fig. 8) and stores it to disk in a RINEX-like format.



```
=============== GRID: GNSS Receiver Implementation on a DSP ===============
 Time since receiver initialization: 16.0 seconds
 PR and ADP measurement time:  310 weeks, 173445.473 secs
                                97134615 samples, 0.898 fractional samples
---------------------------------------------------------------------------
CH   PRN     Dopp       ADP       C/N0      Ik      Qk          PR     Status
             (Hz)     (cycles)   (dB-Hz)                     (meters)
---------------------------------------------------------------------------
 1     1    -19.03      -392.0    47.0    4681    -993          0.00    5
 2     3    430.29      7193.0    47.0   -3467    -381     326034.69    5
 3    11   1233.85     20277.1    47.2   -4786    -128    1057114.04    5
 4    14   -177.31     -2916.0    47.0   -3523     140     105058.25    5
 5    15  -3225.68    -52240.0    47.1    4032    -605    -843787.64    5
 6    16   1984.12     32009.4    46.8   -3853     551   -2062657.90    5
 7    18  -2665.67    -42603.2    47.1    4148     567     -48271.26    5
 8    19    -40.07      -682.7    47.0    3878      35    1688161.64    5
 9    22  -1188.37    -18669.4    47.0   -4176    -198   -2374589.51    5
10    25  -2165.25    -33692.5    47.1   -3517     471   -2003297.19    5
11    30  -3382.47    -51462.6    47.1   -4241    -259    1338195.58    5
12    --   -------  ----------    ----    ----    ----    ----------    -
==========================================================================
Press q to quit
```

Fig. 8. The user interface display on the host computer.

C. Fixed-Point Calculations

The highest performance DSPs, including the ‘C6416, are fixed-point processors, meaning that they provide no hardware support for floating-point operations. Floating point operations can be simulated by the C/C++ compiler, but the resulting loss in efficiency discourages their use in sections of code that must execute quickly.

All but one line of code in the current version of the DSP-based receiver has been implemented using fixed-point operations. The large dynamic range of some variables within the receiver source code complicates the fixed-point implementation, requiring for these variables the use of 64-bit precision. All scaling for fixed-point operations is carried out in base-2 arithmetic except in cases where base 10 is required to avoid large roundoff errors. Base-2 scaling allows the compiler to exploit the DSP’s single-operation barrel shifter, increasing fixed-point efficiency.

D. Memory Use

Reducing memory requirements and properly placing variables within memory are paramount concerns in embedded software design. Memory for the current system is limited to the ‘C6416’s 1024-kB on-chip SRAM and 16 MB of off-chip SDRAM on the DSK. These storage locations are not interchangeable: operations on off-chip data introduce severe computational penalties. Consider: a dot-product operation between two 1000-element, 16-bit-per-element arrays requires a factor of 73 more clock cycles when performed on data stored off-chip as opposed to on-chip; for a 16384-point FFT, there is a 34-fold loss in efficiency using off-chip data.

One deals with memory limitations by (1) reducing the software’s on-chip memory requirements, and (2) by judiciously copying data from off-chip to on-chip memory. As an example of the first approach, consider the memory required by the local carrier replicas, which, for maximum efficiency, are precomputed and stored on-chip. For a Doppler search span of ± 10 kHz, a spacing of 175 Hz (as discussed in Section II), and two-bit quantization at a sampling rate of $F_s = 5.7$ MHz, the carrier replicas require 320 kB of on-chip memory—a considerable fraction of the 1024-kB limit. To reduce this demand, one can compress the carrier replicas by noting that there are no more than 256 different 32-bit words among the bit-packed sign and magnitude, sine and cosine arrays. This fact allows one to encode the 32-bit words as 8-bit elements and expand them very efficiently with a 256-word dictionary. Such an approach reduces the carrier replica’s memory requirement to a modest 81 kB.

As an example of the second approach to dealing with memory limitations, the integer-based (as opposed to bit-packed) carrier and code replicas for FFT-based acquisition are stored off-chip and transferred to on-chip buffers as needed in N-sized chunks. The transfer of data can be effected either by the EDMA or simply by manually copying the data in a `for` loop; the former requires 250 μs per transfer but does not burden the CPU, the latter requires

167 μs of CPU time.

A breakdown of the memory use in the current version of the DSP-based receiver is shown in Fig. 9. The black region at the base of the on-chip memory block indicates the few kB of unallocated memory—a tight fit indeed! Presently, the bit-packed code replicas occupy a relatively large portion (479 kB) of on-chip memory. Real-time generation of these code replicas can reduce their memory requirement to less than 250 kB,[20] but it is anticipated that this will increase correlation time by roughly 30%. Hence, for the current L1 C/A-code receiver, real-time code generation is an unnecessary burden. For the longer L2C and Galileo PRN codes, however, real-time generation will be the only practical option.

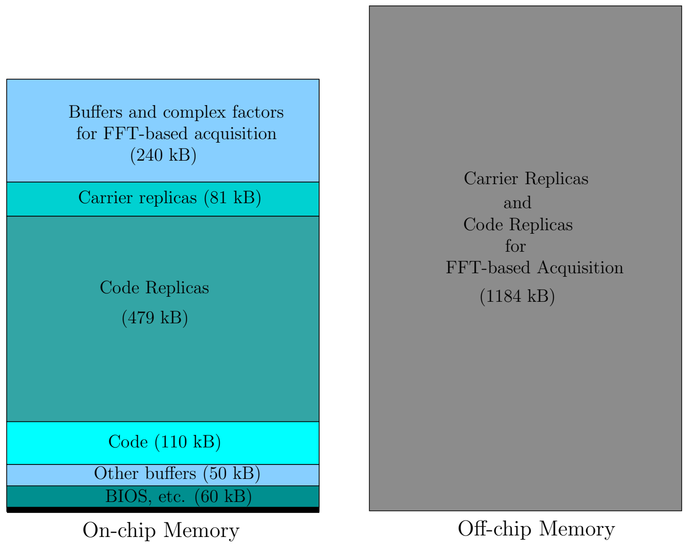


Fig. 9. On-Chip and off-chip memory allocation for the current version of the DSP-based receiver.

V. CODE DEVELOPMENT AND TESTING METHODOLOGY

Figure 10 gives an outline of the code development and testing strategy used in the design of the DSP-based software receiver. First, algorithms are designed and tested in Matlab. These are then implemented in C++ and tested for correctness on the PC using recorded data. The C++ implementation on the PC, with fixed-point arithmetic and loops set up to reduce pipeline overhead, is designed with the DSP target in mind. As needed, `#define` directives are used to isolate the DSP-specific sections of the code. Next, the code is compiled without optimization and loaded onto the DSP. It is then tested for correctness against the PC implementation using identical batches of recorded data. Optimization is then enabled (with compiler options -o3 and -mt) and the code is again tested. This step was found to be necessary when it was discovered that the TI optimizing compiler introduces errors into certain division operations. To prevent this, variables used in such operations are declared `volatile` to protect them from optimization. Finally, the optimized code is run in real-time on live data. Such a pedantic approach to code development and testing is tedious but efficacious.

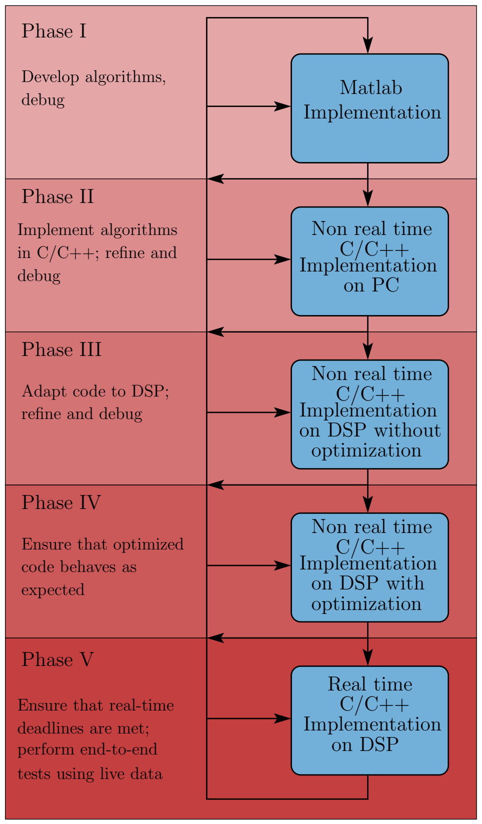

Phase I
Develop algorithms, debug
Matlab Implementation
Phase II
Implement algorithms in C/C++; refine and debug
Non real time C/C++ Implementation on PC
Phase III
Adapt code to DSP; refine and debug
Non real time C/C++ Implementation on DSP without optimization
Phase IV
Ensure that optimized code behaves as expected
Non real time C/C++ Implementation on DSP with optimization
Phase V
Ensure that real-time deadlines are met; perform end-to-end tests using live data
Real time C/C++ Implementation on DSP

Fig. 10. Code development and testing strategy used in the design of the DSP-based receiver.

VI. PERFORMANCE

In the following sections, the DSP-based receiver's performance is assessed in terms of timing benchmarks, sensitivity, tracking in the face of severe vehicle dynamics, and positioning accuracy.

A. Timing Benchmarks

The numbers reported here assume that the target DSP is the TMS320C6416 at 720 MHz with memory allocated as in Fig. 9, sampling frequency $F_s = 5.7$ MHz, 2-bit quantization, and acquisition and tracking of GPS L1 C/A signals only. It is further assumed that all the processes used for benchmarking are run uninterrupted except by the HWIs that increment timers and the circular buffer counter. Three benchmarks are proposed for evaluating software receiver execution speed:

1. Execution time for a search of $N = 8192$ code phase offsets for one noncoherent sum at one test Doppler frequency ($T_c = 1$ ms): $T_1 = 824$ μs.

2. Execution time for one set of 1-ms prompt and EML, in-phase and quadrature correlations: $T_2 = 11.19$ μs.

3. Average execution time for one complete channel update where updates occur at 1-ms intervals: $T_3 = 23.1$ μs.

Benchmark (1) gives a measure of the speed with which acquisition can be performed on a software receiver. One calculates the time required for a complete search for one PRN signal using a search of $N = 8192$ code phase offsets and K noncoherent sums at each of N_f Doppler frequencies (all based on 1-ms coherent integrations) by

$$T_{acq} = 492 + 415N_f + N_f K T_1 \qquad (1)$$

where T_{acq} is given in μs. For a standard search window of $\pm$6 kHz, a search interval of 350 Hz, and $K = 2$ (reliable acquisition down to $C/N_0 = 42$ dB-Hz), $T_{acq} = 72.7$ ms. In other words, one can search all 32 PRNs down to an acquisition threshold of 42 dB-Hz in 2.3 seconds. A deeper search with $K = 41$ for an acquisition threshold of $C/N_0 =$ 33 dB-Hz requires 1.2 seconds per PRN.

Benchmark (2) gives a measure of the execution speed of complex correlations. $T_2 = 11.19$ μs implies that, absent any other overhead, the DSP-based receiver could track floor(1000/11.19) = 89 channels simultaneously.

Benchmark (3) takes both correlation time and tracking loop overhead into account. $T_3 = 23.1$ μs implies that the DSP-based receiver can track floor(1000/23.1) = 43 channels simultaneously. This figure has been verified in practice by tracking 6 live signals and loading the DSP's CPU artificially with an equivalent computational load of 37 more channels. Real-time deadlines were met.

B. Sensitivity

Acquisition and tracking sensitivity were tested using a Spirent GPS simulator. The thresholds calculated for "reliable" acquisition performance (cf. Section II-B.1) were increased by a factor of 1.2 to account for minor correlation peaks. At the target acquisition sensitivity $C/N_0 = 32$ dB-Hz, these elevated thresholds provided the desired $P_{FA} \sim 0.01$, but they naturally decreased the probability of detection to below $P_D = 0.95$. To ensure $P_D \geq 0.95$ required C/N_0 to increase by 1 dB to 33 dB-Hz. Hence, the DSP-based receiver's acquisition threshold using $K = 41$ noncoherent summations is 33 dB-Hz. This threshold was verified in tests with the Spirent simulator.

Acquisition tests were also run under conditions of disparate incoming signal power. It was found that the receiver experienced a dramatic increase in the false alarm rate at the acquisition threshold ($C/N_0 = 33$ dB-Hz) when several strong signals ($\geq$ 49 dB-Hz) were present. In fact, on rare occasions these false alarms lead to bit lock, frequency lock, phase lock, and subframe lock (with parity check) even though the PRN used for tracking was not among the signals present in the incoming data. This behavior is known in multiple-access systems as the near/far problem.[21] It occurs whenever cross-correlation with a strong signal produces correlation peaks that are commensurate with the largest peak produced by the autocorrelation of weak signals. The near/far problem is a challenge for acquisition in indoor environments. Possible strategies for mitigating the near/far problem are discussed in Section VII-A.

Tracking sensitivity is demonstrated in Fig. 11. For this test, the PLL bandwidth was fixed at 8 Hz (i.e., it was not allowed to vary with C/N_0) for comparison against other constant-bandwidth PLLs. Hence, the results of Fig. 11, which demonstrate tracking down to just below $C/N_0 = 25$ dB-Hz, are conservative. Cycle slips were noted (data bit parity failed) near 25 dB-Hz, but the tracking loops did not experience total loss of lock until $C/N_0 < 24$ dB-Hz. No cycle slips occured (data bit parity was maintained) for $C/N_0 \geq 26$ dB-Hz.

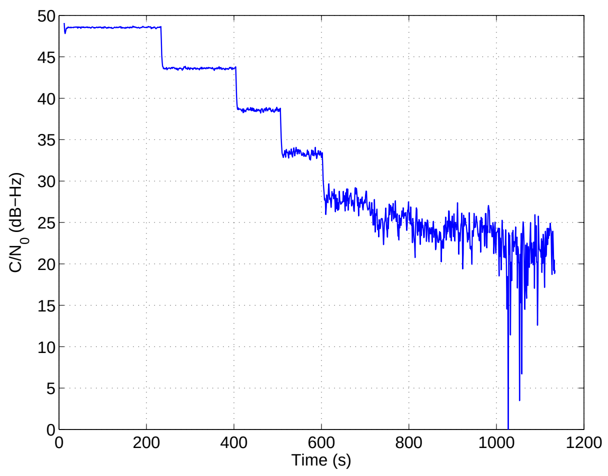


Fig. 11. Tracking sensitivity tests using a graded reduction in the C/N_0 of signals generated by the Spirent simulator.

The DSP-based receiver software was also tested offline with wideband GPS L1 C/A signal data that was recorded in Brazil during severe ionospheric scintillations.[18, 22] Power fades in excess of 25 dB are common in the data set. Figure 12 shows the receiver's C/N_0 estimate over a 20-s window. In no cases studied did the receiver experience a total loss of lock, but it did occasionally experience parity failure after deep power fades, indicating the occurrence of cycle slips. This is consistent with the results of Ref. 18 for similar PLL parameters and structure.

C. Tracking in the Presence of Large Vehicle Dynamics

As a demonstration of the DSP-based receiver's ability to reliably track incoming signals in the presence of large ve-

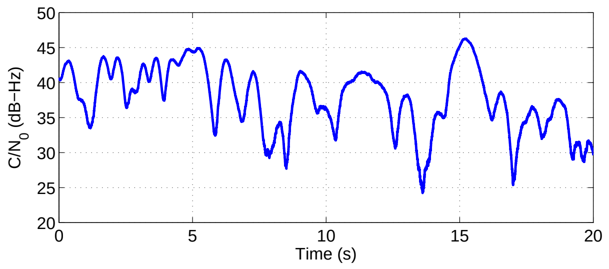


Fig. 12. Offline tracking through ionospheric scintillations using the DSP-based receiver software.

hicle dynamics, a “fighter aircraft” simulation scenario was generated in which the vehicle banks around a circle of 4-km radius at 695 m/s (a little over Mach 2). Signal strength levels were set to a nominal 47 dB-Hz. Such vehicle motion results in 12.3 g's of radial acceleration and large line-of-sight jerk, which severely stresses the carrier tracking loop. For this test, the PLL bandwidth was raised to 15 Hz and the accumulation interval was set to $T_a = 1$ ms. Figure 13 shows the Doppler frequency estimate of the carrier signal from a low-elevation satellite. No cycle slips were detected in the carrier phase measurements.

The “fighter aircraft” test is included here to demonstrate that the DSP-based receiver does not resort to “duty cycle tracking,” a technique used in some software receivers to increase the apparent number of channels that the receiver can simultaneously track. In “duty cycle tracking” the tracking loops only operate on a fraction of the data in a given data window and then “coast” to the beginning of the next data window. This technique limits the bandwidth of the tracking loops and is not suitable for tracking in the presence of large vehicle dynamics.

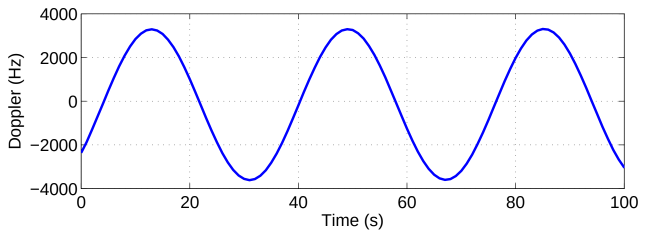


Fig. 13. A low-elevation signal's Doppler frequency estimate during the “fighter aircraft” simulation.

D. Position Accuracy

The accuracy of the DSP-based receiver's pseudorange observables was tested by generating a set of navigation solutions off-line using the RINEX-like files produced by the receiver. Results are presented in Fig. 14; positioning errors are consistent with what one would expect for the Standard Positioning Service GPS signal (cf. Ref. 23, p. 217).

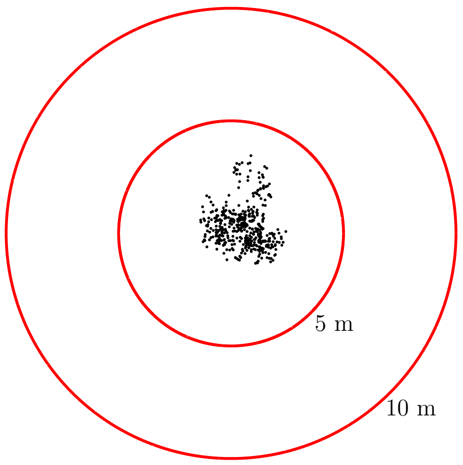


Fig. 14. Horizontal code-phase position accuracy for the DSP-based receiver over a 45-second interval using live data taken from atop Rhodes Hall, Cornell University. The center of the plot indicates the true antenna position as defined by a 3-day average using a high-quality single-frequency commercial GPS receiver.

VII. PROSPECTS

The foregoing performance results make it clear that an embedded software receiver based on a high-performance DSP can compete with standard application-specific integrated circuit (ASIC) chipsets over a broad range of relevant performance measures. These results and others that will surely follow will help to dispel skepticism within the GNSS community about whether software receivers can “succeed as as generic high-end receivers or if they can penetrate the embedded market.”[6]

In this section, the prospects of the DSP-based receiver are discussed, first by considering uses for the receiver's ample throughput (beside simply performing acquisition in the background), and then by considering possible alternatives to the TMS320C6416 DSP.

A. Cross-Correlation Mitigation

Techniques for mitigating the cross-correlations caused by the near/far problem, which was introduced in Section VI-B, have been presented in the communications literature.[21] These involve canceling the strong (“near”) signals by subtracting them successively from the incoming RF front end data. For GNSS chipsets based on hardware correlators, this approach requires the inclusion of specialized cancellation hardware within the hardware correlator chip. For a software receiver with sufficient throughput, it requires a software subroutine that takes as arguments the incoming data stream and the tracking parameters of a strong signal already being tracked. The subroutine generates a replica signal whose phase, frequency, and amplitude are aligned with those of the interfering strong signal and subtracts the replica from the data stream.

The above cancellation method mitigates large cross correlations that arise during acquisition. For large cross correlations that arise during tracking, a different approach is required because the bit-wise parallel correlation method does not lend itself to signal cancellation by subtraction. Instead, spare channels can be dedicated to performing cross-correlation calculations between the code replicas of the strong signal and the weak signal being tracked. The result is subtracted from the weak signal's (I, Q) vector to perform a cross-correlation "fix-up" after each accumulation.

Efforts are underway to implement both of the foregoing cross-correlation mitigation strategies on the DSP-based receiver.

B. Incorporating other Signals

B.1 L2C

The bandwidth of the L2C signal is, like the L1 C/A signal, approximately 2 MHz. Hence, no changes in sampling rate would be required to incorporate L2C onto the current platform. Indeed, Ref. 3 demonstrates an L1 C/A + L2C software receiver based on two GP2015 front-ends sampling at $F_s = 5.7$ MHz. For use on the DSP, the L1 C/A and the L2C data streams from the two front-ends would need to be interleaved because the 'C6416 has only three McBSPs, one of which is currently dedicated to L1 C/A sign bits and another to L1 C/A magnitude bits. Each McBSP supports data rates up to 125 Mbps[24]—more than adequate for interleaved signals.

Bit-packed L2C medium (CM) and long (CL) codes will not fit in on-chip memory and must be generated in real time,[20] adding approximately 30% to the cost of correlation. Moreover, the CM and CL codes must be tracked independently because of the data bits modulated onto the CM code. Hence, each L2C channel update will be approximately 2.3 times as expensive as the current L1 C/A channel updates.

To work within the limitations of the DSP's FFT routines, L2C acquisition will either have to be aided by L1 C/A acquisition or be carried out using the techniques introduced in Ref. 25.

B.2 Galileo L1 BOC(1,1)

The Galileo L1 BOC(1,1) signal's wider bandwidth (~ 4 MHz) will require roughly double the sampling rate of the current implementation. A front-end chip with a variable-bandwidth low-pass filter that is already available for the L1 C/A signal can be used as an L1 C/A + Galileo L1 BOC(1,1) front-end.[26] The 4-ms PRN codes at the higher sampling rate will not fit into on-chip memory; hence, as for L2C, these must be generated in real time,[20] adding approximately 30% to the cost of correlation at the higher sampling rate. Accordingly, each Galileo channel update will be approximately 1.6 times as expensive as the current L1 C/A channel updates. Galileo acquisition will also have to proceed along the lines of Ref. 25.

As a demonstration of the ease with which the DSP-based software receiver's code can be adapted to new GNSS signals, the authors conducted an experiment in flexibility. Starting with the L1 C/A framework (admittedly designed with other GNSS signals in mind), the authors converted the code to a Galileo L1 BOC(1,1) receiver that could operate off-line on wide bandwidth (41 MHz) recorded data that included the BOC(1,1) signal transmitted by GIOVE-A. The conversion took less than 3 hours. Figure 15 shows the results of tracking the GIOVE-A L1-B signal. The PRN codes required to track GIOVE-A were provided by Cornell's GNSS research group.

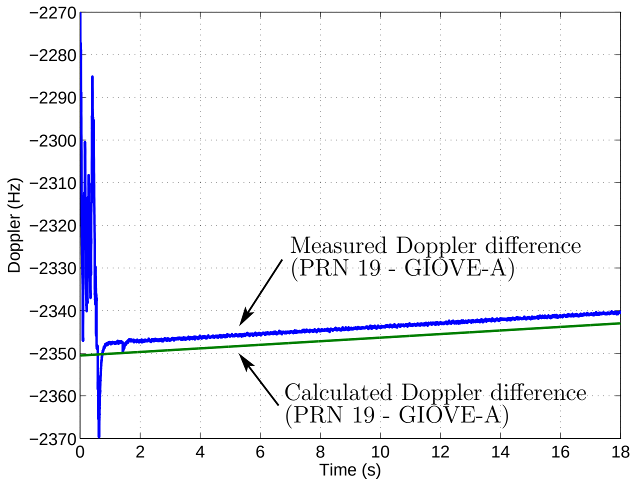


Fig. 15. Results of off-line tracking the GIOVE-A signal using recorded data. Shown in blue is the difference between the measured Doppler time histories of PRN 19 and GIOVE-A. The difference calculation removes the receiver clock frequency bias. Shown in green is the same difference calculated using the broadcast GPS ephemerides for PRN 19 and the NORAD elements for GIOVE-A. The residual 3-Hz difference between the two plots is likely a result of errors in the NORAD elements.

C. Alternatives to the TMSC6416

Trends in DSP design look promising for DSP-based GNSS software receivers. There now exist at least two DSPs more capable than the 'C6416. These and the 'C6416 are listed below, along with the authors' projections of what could practically be implemented on each.

- TI's TMS320C6416 (current target)
  - 720 MHz, 1 MB of internal memory, \$120
  - 13 L1 C/A + 12 L2C channels ($F_s = 5.7$ MHz)
- TI's TMS320C6455
  - 1 GHz, 2 MB of internal memory, new C64+ core, \$200

– 10 L1 C/A + 8 L2C + 10 Galileo L1 BOC(1,1) channels ($F_s = 10$ MHz)

- Freescale's MSC8144
  - quad 1-GHz processors, 10 MB internal memory, \$230
  - 12 L1 C/A + 12 L2C + 12 Galileo L1 BOC(1,1) channels ($F_s = 10$ MHz)

One might also entertain thoughts of implementing a GNSS receiver on a smaller, more economical DSP:

- TI's TMS320C6414
  - 600 MHz, 1 MB of internal memory, \$80
  - 11 L1 C/A + 9 L2C channels ($F_s = 5.7$ MHz)

VIII. SUMMARY

- Status: An L1 C/A-code GPS receiver has been implemented on a Texas Instruments TMS320C6416 DSP. The receiver can track up to 43 real-time channels at a 2-bit sampling rate of 5.7 MHz. Its FFT-based acquisition strategy is capable of searching through 32 nominal-strength GPS signals in just over 2 seconds. By increasing the number of noncoherent sums that it uses, its acquisition threshold can be made as low as $C/N_0 = 33$ dB-Hz. Reliable tracking down to $C/N_0 = 25$ dB-Hz has been demonstrated. The receiver produces pseudoranges and proper carrier phase measurements. Its code can be readily adapted for use with other GNSS signals; to demonstrate this, the code was used to do non-real-time tracking of the Galileo GIOVE-A signal.
- Challenges: For high performance, the code implementation must employ fixed-point arithmetic, configure loops properly for pipelining, and judiciously allocate the limited on-chip memory.
- Prospects: High-performance DSPs appear to be promising targets for dual-frequency science-grade embedded GNSS software receivers. The fastest DSP on the market today appears to be capable of handling 12 L1 C/A + 12 L2C + 12 Galileo L1 BOC(1,1) channels at a 2-bit sampling rate of 10 MHz.

ACKNOWLEDGMENTS

This work has been supported in part by the NASA Office of Space Science through grant No. NNX06AC34G with Madhulika Guhathakurta as grant monitor. Support for the weak signal acquisition and tracking aspects of this work has been provided by NASA Cooperative Agreement NCC5-722 with grant monitor Michael C. Moreau.

References

[1] Akos, D. M., Normark, P., Hansson, A., Rosenlind, A., Stahlberg, C., and Svensson, F., "Global Positioning System Software Receiver (gpSrx) Implementation in Low Cost/Power Programmable Processors," *Proc. 2001 ION GPS Conf.*, Institute of Navigation, Salt Lake City, UT, September 2001, pp. 2851–2858.

[2] Akos, D. M., Normark, P., Enge, P., Hansson, A., and Rosenlind, A., "Real-Time GPS Software Radio Receiver," *Proc. 2001 ION NTM*, Institute of Navigation, Long Beach, CA, January 2001, pp. 809–816.

[3] Ledvina, B. M., Psiaki, M. L., Powell, S. P., and Kintner, Jr., P. M., "Real-Time Software Receiver Tracking of GPS L2 Civilian Signals using a Hardware Simulator," *Proc. 2005 ION GNSS Conf.*, Institute of Navigation, Long Beach, CA, September 2005.

[4] Ledvina, B. M., Cerruti, A. P., Psiaki, M. L., Powell, S. P., and Kintner, Jr., P. M., "Performance Tests of a 12-Channel Real-Time GPS L1 Software Receiver," *Proc. 2003 ION GPS Conf.*, Institute of Navigation, Portland, OR, 2003.

[5] Dovis, F., Spelat, M., Mulassano, P., and Leone, C., "On the Tracking Performance of a Galileo/GPS Receiver Based on Hybrid FPGA/DSP Board," *Proc. 2005 ION GNSS Conf.*, Institute of Navigation, Long Beach, CA, September 2005.

[6] Won, J.-H., Pany, T., and Hein, G. W., "GNSS Software Defined Radio," *Inside GNSS*, Vol. 1, No. 5, July 2006, pp. 48–56.

[7] Leveson, I., "Benefits of the New GPS Civil Signal," *Inside GNSS*, Vol. 1, No. 5, July 2006, pp. 42–47,56.

[8] Hein, G. W., Pany, T., Wallner, S., and Won, J.-H., "Platforms for a Future GNSS Receiver," *Inside GNSS*, Vol. 1, No. 2, March 2006, pp. 56–62.

[9] Lapsley, P., Bier, J., Shoham, A., and Lee, E. A., *DSP Processor Fundamentals*, IEEE Press Series on Signal Processing, IEEE Press, New York, 1997.

[10] Bier, J., "The Evolution of DSP Processors," http://www.bdti.com/articles/slides.evolution.pdf, November 1997.

[11] van Nee, D. J. R. and Coenen, A. J. R. M., "New Fast GPS Code Acquisition Technique Using FFT," *Electronics Letters*, Vol. 27, No. 2, January 1991, pp. 158–160.

[12] "A BDTI Analysis of the Texas Instruments TMS320C64x," Tech. rep., Berkeley Design Technology, Inc., www.bdti.com, 2004.

[13] Texas Instruments, www.ti.com, *TMS320C6414T/15T/16T Fixed-Point Digital Signal Processors (SPRS226J)*, July 2006.

[14] Texas Instruments, www.ti.com, *TMS320C6000 Optimizing Compiler Users Guide (SPRU187L)*, May 2004.

[15] Ledvina, B. M., Psiaki, M. L., Powell, S. P., and Kintner, Jr., P. M., "Bit-Wise Parallel Algorithms for Efficient Software Correlation Applied to a GPS Software Receiver," *IEEE Transactions on Wireless Communications*, Vol. 3, No. 5, September 2004.

[16] Texas Instruments, www.ti.com, *TMS320C64x DSP Library Programmers Reference (SPRU565B)*, October 2003.

[17] Van Dierendonck, A. J., *Global Positioning System: Theory and Applications*, chap. 8: GPS Receivers, American Institute of Aeronautics and Astronautics, Washington, D.C., 1996, pp. 329–407.

[18] Humphreys, T. E., Psiaki, M. L., Ledvina, B. M., and Kintner, Jr., P. M., "GPS Carrier Tracking Loop Performance in the Presence of Ionospheric Scintillations," *Proc. 2005 ION GNSS Conf.*, Institute of Navigation, Long Beach, CA, September 2005.

[19] Texas Instruments, www.ti.com, *TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide (SPRU580C)*, March 2004.

[20] Psiaki, M. L., "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.

[21] Duel-Hallen, A., Holtzman, J., and Zvonar, Z., "Multiuser Detection for CDMA Systems," *IEEE Personal Communications*, April 1995, pp. 46–58.

[22] Humphreys, T. E., Ledvina, B. M., Psiaki, M. L., and Kintner, P. M., "Analysis of Ionospheric Scintillations using Wideband GPS L1 C/A Signal Data," *Proc. 2004 ION GNSS Conf.*, Institute of Navigation, Long Beach, California, 2004, pp. 399–407.

[23] Misra, P., *Global Positioning System, Signals, Measurements, and Performance*, Ganga-Jumana Press, Lincoln, Massachusetts, 2006.

[24] Anjanaiah, S. and Soteriou, V., "Using the TMS320C6000 McBSP as a High Speed Communication Port (SPRA455A)," Tech. rep., Texas Instruments, www.ti.com, August 2001.

[25] Psiaki, M. L., "FFT-Based Acquisition of GPS L2 Civilian CM and CL Signals," *Proc. 2004 ION GNSS Conf.*, Institute of Navigation, Long Beach, CA, September 2004, pp. 457–473.

[26] Maxim, www.maxim-ic.com, *Integrated L1-band GPS Receiver*, January 2005.