OpenSource GPS

A Hardware/Software Platform for Learning GPS: Part II, Software

Clifford Kelley, OpenSource GPS, and **Douglas Baker,** GPS Creations

THIS MONTH'S COLUMN FEATURES THE SECOND INSTALLMENT OF A TWO-PART ARTICLE ABOUT THE OpenSource GPS project — an economical hardware and software platform developed specifically to allow students access to the inner workings of a GPS receiver. In Part I, we looked at the hardware component, built around the Zarlink GP2015/GP2021 chipset. In Part II, project leaders Clifford Kelley and Douglas Baker overview the project software and present results of static and kinematic testing of the OpenSource GPS platform. — R.B.L.

As we mentioned in last month's "Innovation" column, the OpenSource GPS Project began in 1995 as a noncommercial alternative to the pioneering GPS Builder product introduced by GEC Plessey Semiconductor in Swindon, United Kingdom. (GEC Plessey was acquired by Mitel Semiconductor, Ottawa, Canada, in 1998 and reborn in 2001 as Zarlink Semiconductor.) Based on the same family of GPS chipsets, OpenSource GPS provides students an opportunity to learn about GPS by experimenting with the hardware and software of a GPS receiver.

Last month, we discussed the hardware platform and its evolution. The current version uses the Zarlink GP2015 front-end chip and the GP2021 12-channel correlator chip. A radio-frequency board containing both chips but without baseband signal processing is available as a daughter board that plugs into an Industry Standard Architecture (ISA) card or a Peripheral Component Interconnect (PCI) card for installation in a PC. This month, we discuss the project software running on a PC, which communicates with and controls the actions of the correlator chip and processes the chip's correlator measurements.

Software

OpenSource GPS is a C program written in C/C++ and compiles under Borland versions 4.5 and later, Microsoft Visual C++ 1.5, and DJ's GNU Programming Platform (DJGPP), a free C++ compiler. The software functionality extends from satellite acquisition and tracking to the decoding of the navigation message and to producing raw pseudorange and delta-pseudorange (carrier-phase) measurements.

The software uses a single interrupt routine to handle the tracking loops and find the navigation message. All other functions are handled using a polling method triggered by flags from the interrupt routine. To reduce connections to the correlator chip, the PC clock timing (interrupt [int] 0) is taken over and is modified to provide an interrupt every 500–900 microseconds to check each channel for the in-phase (I) and quadrature (Q) correlation data. (See the accompanying sidebar **"Minding Your *I*s and *Q*s"** on page 56 for an introduction to I and Q signal processing.)

Figure 1 is a file-structure diagram of the software. The program is written in three sections: The main code runs the receiver and provides functions that deal with communication with the correlator chipset, navigation fixes, and almanac and ephemeris data management. The second section is the matrix library. The third is the serial-port library. The only pure input file used is the rcvr_par.dat file, which contains constants used by the receiver such as tracking loop constants for code, carrier for pull-in and tracking, and flags for various outputs. The input/output files store almanac and ephemeris data that are read at the start of the program and updated when the program exits. The output files record data for later analysis.

Each channel provides four correlation counts. The prompt and dither (either ahead of or behind the prompt correlator by 1/2 chip) both have I and Q correlators. By forming a root sum square (rss) of the in-phase and quadrature values, the software can determine if they are lined up with the psuedorange noise (PRN) code. The in-phase and quadrature correlator values indicate the phase of the signal. To track the signal, the software does not move the correlators directly, it just speeds up or slows down the carrier and code digitally controlled oscillators (DCOs) to keep the channel locked in code and carrier phase.

As mentioned, the correlator read timing of the receiver is set by taking over the PC's interrupt instruction, int 0, which normally is used to keep the computer real-time clock up to date. An Intel 8254 counter/timer chip normally interrupts the computer approximately every 50 milliseconds. The software replaces this interrupt routine with its own (GPS_interrupt) and sets the interrupt time to approximately 500 microseconds. Whenever the interrupt occurs, the software checks the channel data register to see which channels have dumped correlation data. The correlator data are read and a check is performed to see if a 99.9999-milliseconds tic has occurred. If it has, the measurement data are stored. Each channel is processed based on the state of the channel.

States. State 1 is the acquisition state.

It performs the code and frequency (Doppler) search to find a high correlation peak. If a peak in either the prompt or delay (dither) correlator (rss of in-phase and quadrature outputs) is greater than the set threshold, it switches to State 2. When in warm- or cold-start mode, the Doppler frequency is centered on the expected value. The search is conducted with two loops; the inner loop is the code search, and the outer is the frequency search. The code search pattern simply is a slewing of the PRN code generator by adding an extra chip of delay after each correlation sample. Thus, after 1023 samples, every 1/2 chip in code space has been searched. The frequency search pattern checks either side of the expected frequency in integer multiples of the Doppler search range; that is, (0, +1, –1, +2, –2, . . .) × 200 Hz. The maximum range depends on what the receiver is doing. The nominal value is for a warm start. The value is larger if attempting a cold start and smaller if the receiver has a navigation fix. Samples are taken every millisecond, which allows each frequency bin to be searched in approximately 1 second, if the software does not go into State 2.

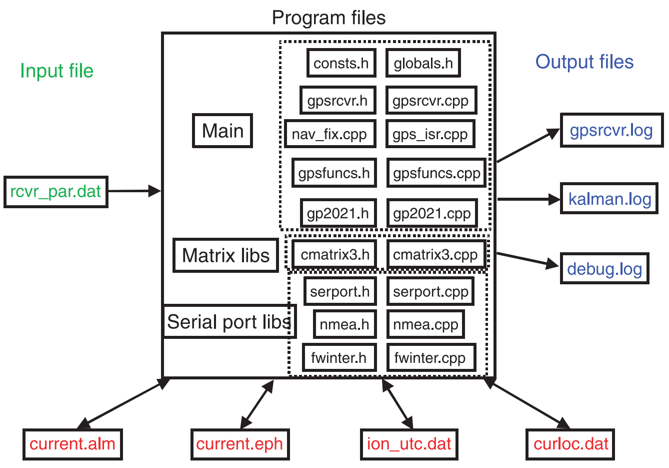

FIGURE 1 OpenSource software source code and data file structure

State 2 is the confirmation state. The confirmation state stops the search and dwells at the code offset and Doppler frequency at which the high correlation peak was found in State 1 to confirm the presence of the signal to reduce the false alarm rate. If *n* of *m* samples (for example, 8 of 10) are greater than the threshold, the software switches to State 3.

State 3 is the pull-in state. In this stage, the signal has been confirmed, but the frequency may be off by as much as 500 Hz. The pull-in state attempts to start tracking the signal to pull the frequency in close enough so that the carrier phase can be tracked. In addition, the code loop is closed after 2 milliseconds and the carrier loop is closed after 5 milliseconds to reduce start-up transients. This state is enabled for approximately 1500 milliseconds. During the last 500 milliseconds of this time interval, the C/N$_0$ and phase errors are measured, and the program attempts to synchronize onto the edge of a data bit. If the software confirms carrier and code tracking, it switches to State 4. Because the frequency is likely to be very far off, the receiver uses a combination of a frequency-locked loop (FLL) and a phase-locked loop (PLL). As the time progresses in the state, the effect of the FLL is decreased so that the PLL is dominant at the end of the state.

State 4 is the normal tracking state. The tracking loops are aligned with and integrate during a data bit (20 milliseconds) to track code and 1 millisecond to track carrier phase. The data message is recorded and the time is synchronized to the time of week (TOW) of the data message. To reduce the code-tracking noise, the code-tracking loop is aided by the carrier-tracking loop.

The software interprets the navigation message and computes a navigation fix in position, velocity, and time according to the algorithms specified in the *Navstar GPS Space Segment/Navigation User Interfaces* interface control document (ICD-GPS-200), now superseded by IS-GPS-200D. To provide accuracies on the order of a few meters, all of the corrections must be made. These include the Sagnac effect, relativistic effects, and troposphere and ionosphere models.

Updates. Since the original code was developed, it has been restructured into library components to make it more maintainable and extensible. One added library outputs National Marine Electronics Association (NMEA) formatted messages on an RS-232 serial port. This function allows another computer to incorporate the position data for other mapping and graphical displays.

A Kalman filter was added, made much easier by using a matrix and vector package developed for the project. Upgrades to the software include continuous checks for new almanac and ephemeris messages and new integrity checking for data-bit synchronization in the pull-in routine.

Additional effort has gone into providing documentation. One of the most-difficult tasks in any software project is to obtain good documentation. Although several good books describe GPS principles, none goes into the detail required to understand the complex interaction of acquisition, tracking, and measurement generation in a real-time program. Although the project documentation necessarily is quite specific to the chipset used, it also touches on general issues as well. This documentation is a work in progress and is available in Portable Document Format (PDF) on the project Web site. Another program called regtest has been written to test the basic communications and functions of the chipset. Regtest checks the data bus and address bus connections between a computer and the GP2021 and reports any problems.

Displays. Figure 2 shows the first and main display page when the program is run. The top of the screen provides general data such as software version, inter-

```
         OpenSource GPS Software Version 1.18 PCI
Sun Sep 04 10:17:54 2005
TOW   62274
Meas time 62249.999973 error 0.000000 delta 0.000000
   latitude    longitude         HAE     clock error (ppm)
    33:41:50.63  -118:21:31.91     292.32 0.288399
 Speed      Heading        TIC_dt
 0.014914    -158.446599    0.999999

Tracking  6 status 4 almanac valid 0 gps week 315 alm_page 51
 ch prn state n_freq az   el doppler t_count n_frame sfid ura page missed CNo
  0  1    4    1      51    42    -1500     44      1    5    0    0    1   47.2
  1 11    4    3    -131    34    -2160     44      1    5    0    0    1   45.7
  2 14    4    2      73    12    -2305    975      1    0    0    0    1   39.4
  3 16    4    3     141    26     2883     44      1    5    1    0    1   49.9
  4 20    4    3     -40    53     2133     44      1    5    0   51    1   50.2
  5 23    4    1     -73    32     2823     44      1    5    0    0    1   49.3
  6 24    4    6     -66    29     1623     44      0    5    0    0    1   49.1
  7 25    4    2      44    53    -2116     44      1    5    0    0    1   40.5
  8  0    0    0       0     0        0      0      0    0    0    0    0    0.0
  9  0    0    0       0     0        0      0      0    0    0    0    0    0.0
 10  0    0    0       0     0        0      0      0    0    0    0    0    0.0
 11  0    0    0       0     0        0      0      0    0    0    0    0    0.0
 GDOP= 4.452  HDOP= 1.278    VDOP= 3,290  TDOP= 2.117
```

FIGURE 2 OpenSource GPS display page 1

face version (ISA or PCI), position, and velocity fix information and stays the same for all of the display pages. The lower part of the screen provides data for each channel such as PRN code number assignment, the tracking state, search frequency, and azimuth and elevation angle to the satellites. In addition, the Doppler frequency bit count, number of frame numbers recorded, subframe identification, user range accuracy, almanac page, number of missed correlations, and C/N_0 are provided.

Four additional screen pages provide further information about the operation

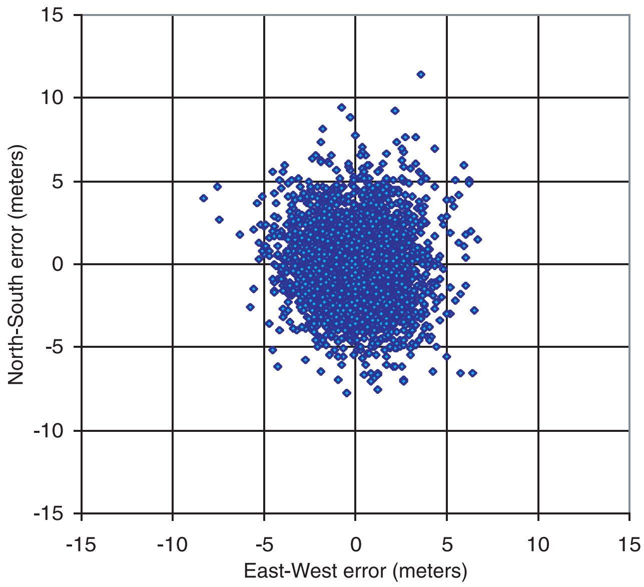

FIGURE 3 Horizontal position error scatter plot

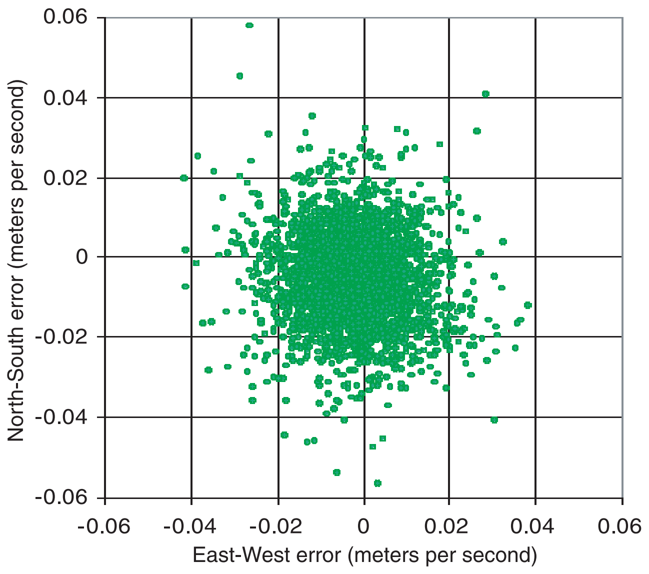

FIGURE 4 Horizontal velocity error scatter plot

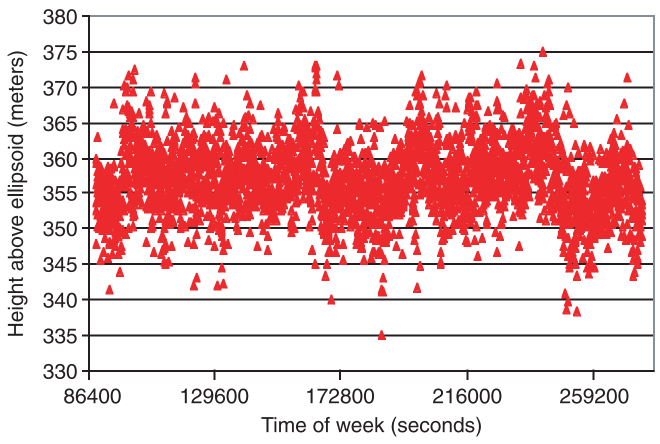

FIGURE 5 Vertical position scatter plot

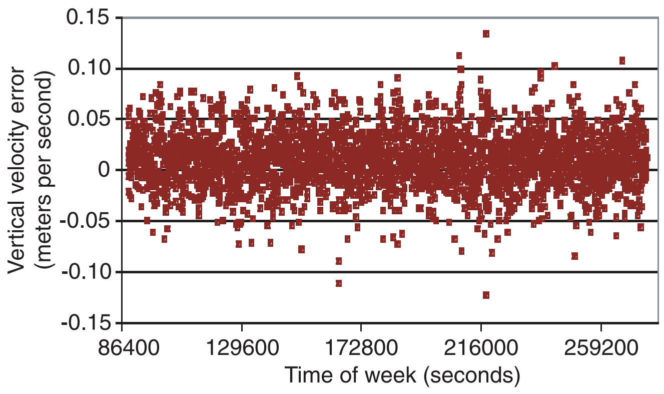

FIGURE 6 Vertical velocity scatter plot

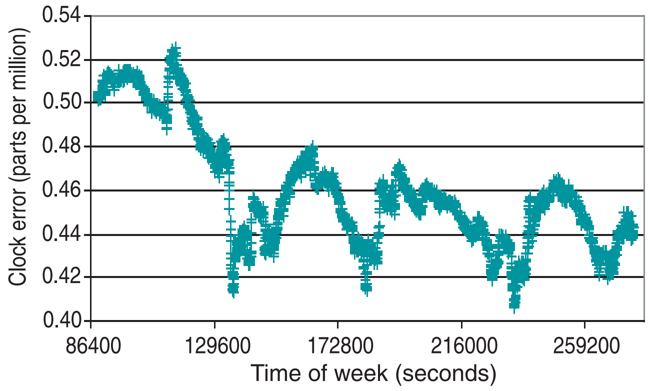

FIGURE 7 Clock frequency error plot

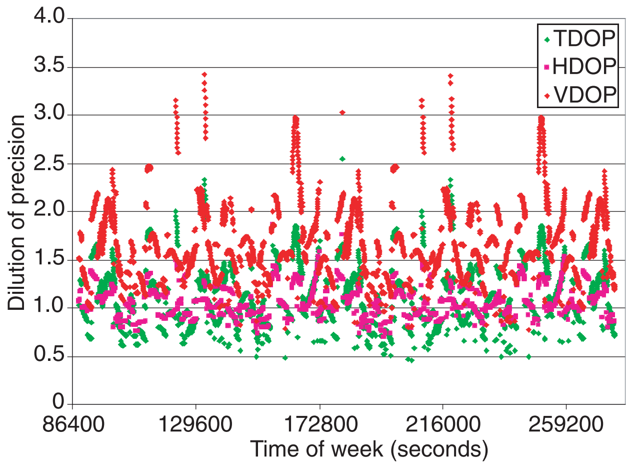

FIGURE 8 DOP levels during the static test

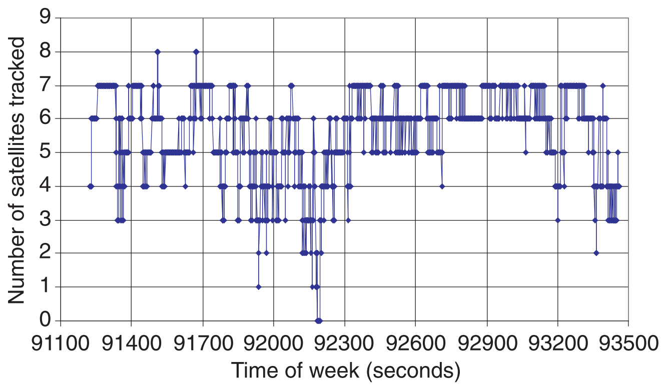

FIGURE 9 Number of satellites tracked by OpenSource GPS during automobile trajectory

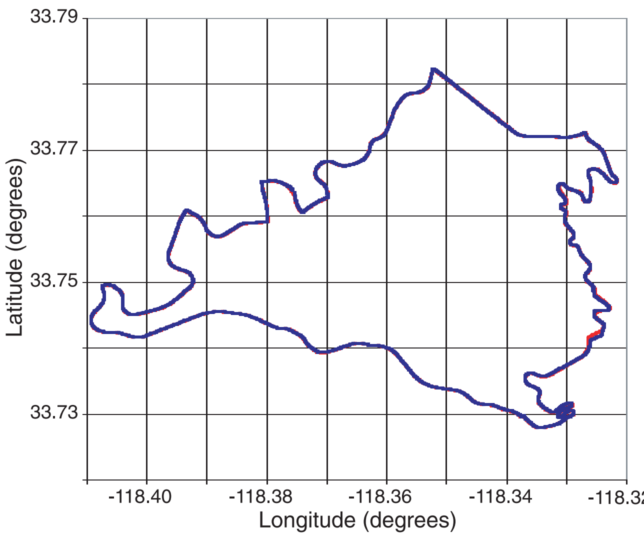

FIGURE 10 Automobile trajectory: OpenSource GPS (dark blue) versus a commercial receiver Navman Jupiter (red). The east–west extent of this plot is approximately 10 kilometers.

of the receiver and the software. Page 2 provides data about each satellites' telemetry, TOW navigation message words, and other parameters used to determine if the measurements can be used.

Page 3 adds information about the size of the corrections for tropospheric and ionospheric delays that are being used.

Page 4 adds information about the pseudorange and delta-pseudorange data for each satellite.

Page 5 adds information about the parity errors detected in the messages from each satellite.

Test Results

We have performed both static and kinematic tests with the software to determine how well it works compared with theory and with the operation of commercial receivers. **Figures 3 – 7** show the results of a static test run during a 52-hour period with data recorded at 1-minute intervals. This data set was obtained without the use of a Kalman filter. The circular error probable is approximately 2.4 meters, and the spherical error probable is approximately 4.5 meters. **Figure 8** shows the dilution of precision (DOP) values obtained during the test. The performance appears to be comparable to that of an unaided commercial receiver.

The kinematic test consisted of a course around the winding, hilly streets of the Palos Verdes Peninsula in southern Los Angeles County. We used a Kalman filter to process the raw measurements. **Figure 9** shows the number of satellites tracked during the test. **Figure 10** is a comparison of the trajectory obtained with OpenSource GPS (dark blue) and a commercial receiver, a Navman Jupiter (red). We observed noticeable differences in the two trajectories only in two places where the number of satellites tracked was reduced to 1 or 0 satellites.

The Future

Obviously no software is perfect, especially when written in a developer's spare time and without access to sophisticated test equipment. It appears to work pretty well; however, several areas have problems and improvements are needed in other areas:

Minding Your *I*s and *Q*s

A radio signal may be represented, in general, as

$$x(t) = A\cos(\omega t + \varphi)$$

where *A* is the instantaneous amplitude, ω is the instantaneous carrier frequency in radians per second, and φ is the instantaneous phase offset. Depending on the modulation type, the amplitude, frequency, and phase can be functions of time.

How can we determine the amplitude and phase of an unknown signal if we only know its frequency? A common approach starts by multiplying the signal by a pair of reference signals

$$\cos(\omega t) \text{ and } \cos(\omega t + 90°) = -\sin(\omega t)$$

These reference signals are 90° out of phase with each other. In other words, they are orthogonal or in phase quadrature. The resulting signals are

$$I = A\cos(\omega t + \varphi)\cos(\omega t)$$
$$Q = -A\cos(\omega t + \varphi)\cos(\omega t)$$

Using a bit of trigonometry, we find that

$$I = \frac{1}{2}A\left[\cos(\omega t + \varphi - \omega t) + \cos(\omega t + \varphi + \omega t)\right]$$
$$= \frac{1}{2}A\left[\cos(\varphi) + \cos(2\omega t + \varphi)\right]$$
$$Q = \frac{1}{2}A\left[\sin(\omega t + \varphi - \omega t) - \sin(\omega t + \varphi + \omega t)\right]$$
$$= \frac{1}{2}A\left[\sin(\varphi) - \sin(2\omega t + \varphi)\right]$$

with the signal products consisting of a zero frequency (direct current) component and one with a frequency equal to twice the signal frequency. If these signals are low-pass filtered, we obtain

$$I = \frac{1}{2}A\cos\varphi$$
$$Q = \frac{1}{2}A\sin\varphi$$

I and *Q* are known as the in-phase and quadrature-phase (or quadra-phase) signal components, respectively because *I* is maximized when the unknown signal is phase aligned with cos(ω*t*) and *Q* is maximized when the unknown signal is phase aligned with −sin(ω*t*).

The amplitude of the unknown signal can then be determined using the vector magnitude of the *I* and *Q* components:

$$A = 2\sqrt{I^2 + Q^2}$$

and the phase can be determined using the four-quadrant arctangent of the ratio of the *Q* and *I* components:

$$\varphi = \tan^{-1}\left(\frac{Q}{I}\right)$$

If a radio receiver provides both in-phase and quadrature-phase local oscillator signals, then an amplitude modulated (AM) radio signal can be demodulated using the vector magnitude of the filtered *I* and *Q* intermediate-frequency products. In fact, signals using any of the common modulation techniques can be demodulated by manipulating the *I* and *Q* products. The approach works whether using analog or digitized signals.

In a digital GPS receiver, after suitable down conversion of the received signal and digitization, in-phase and quadrature-phase sampled data are produced using *I* and *Q* replica carriers (including carrier Doppler shift) from a numerical (digitally) controlled oscillator. The *I* and *Q* signals are correlated separately with early, prompt, and late replica pseudorandom noise codes (plus code Doppler) synthesized by the code generator. Just two (prompt and dithered) replicas can be used if the dithered replica can be selected to be early or late. After integration of the resulting signals, baseband processing occurs using code and carrier tracking loop discriminators and loop filters to provide correction signals for the code and phase tracking loops and to provide measurements of the replica code phase and replica carrier Doppler phase, which can be converted into pseudorange and carrier-phase measurements for positioning, navigation, and timing solutions.

A transmitted radio signal itself may include separate *I* and *Q* components. For example, the GPS L1 signal is modulated with the C/A-code and the navigation message on the in-phase channel and the P(Y)-code and the navigation message on the quadrature channel. — R.B.L.

■ The acquisition/pull-in steps integrate during 1 millisecond. A longer integration time or variable integration time might be desirable.

■ The tracking loops currently integrate during 20 milliseconds for code and 1 millisecond for carrier phase. Again, a longer integration time or variable integration time could be desirable.

■ Lastly, the tracking loops have not been optimized and could be greatly improved.

Since the presentation of the paper on which this article is based at ION GNSS 2005, work has already begun on version 2.00 of the software in which the code has been restructured to merge the DOS version with Linux. Although it is not yet ready for release as a package, it can be found at SourceForge in the CVS section of project OSGPS. See the OpenSource GPS project Web site, *www.home.earthlink.net/~cwkelley/* for access information. As of early January 2006, version 2.00 appears to be working well with minor debugging needed.

Comments and suggestions are most

welcome. Please direct them to Clifford Kelley at cwkelley@earthlink.net or to Douglas Baker at dbaker@gpscreations.com.

Further Reading

See Part I of this article in the January 2006 issue of *GPS World.*

Acknowledgments

The authors would like to acknowledge the encouragement of Jay Farrell at University of California, Riverside; the interest of Chris Rizos and Jinling Wang from the University of New South Wales, Australia; and the contributions from the many individuals who have collaborated on the project. Additional OpenSource GPS projects have been set up by:

- Georg Beyerle, Opengps running under RTAI Linux using a PCI interface, *www.gfz-potsdam.de/pb1/staff/gbeyerle/opengps*
- Stephan Esterhuizen, OpenGPS at the University of Colorado, Boulder running under RTAI Linux, *http://ccar.colorado.edu/opengps/doc/*
- Yu Lu, USBGPS, running under MS Windows 2000/XP using a USB interface, *www.ee.ucr.edu/~crlab/usbgps/.*

A master list of known OpenSource GPS and related projects is maintained at Portland State University, *http://gps.psas.pdx.edu/OpenGnssProjects.*

This article is based on the paper "OpenSource GPS: Open Source Software for Learning about GPS" by C. Kelley and D. Baker, presented at ION GNSS 2005, the 18th International Technical Meeting of the Satellite Division of The Institute of Navigation, Long Beach, California, September 21–25, 2005. ⊕

CLIFFORD KELLEY holds a B.S. in physics, a B.S. in electrical engineering, an M.S. in systems engineering, and a Ph.D. in industrial and systems engineering, all from the University of Southern California, Los Angeles. He has been working on GPS software since 1995 and began offering it as OpenSource GPS in 2001.

DOUGLAS BAKER has a B.S. in electrical engineering from Vanderbilt University, Nashville, Tennessee, and has been actively working with GPS hardware design since 1987.

"Innovation" is a regular column featuring discussions about recent advances in GPS technology and its applications as well as the fundamentals of GPS positioning. The column is coordinated by **RICHARD LANGLEY** of the Department of Geodesy and Geomatics Engineering at the University of New Brunswick, who appreciates receiving your comments and topic suggestions. To contact him, see the "Columnists" section on page 6 of this issue.

INDEX

ADVERTISERS

COMPANIES