Open Source Software for Commercial Off-the-Shelf GPS Receivers

Andrew Greenberg, *Portland State Aerospace Society*
Takuji Ebinuma, *Mitsubishi Electric Corporation, Kamakura Works*

BIOGRAPHY

Andrew Greenberg has a B.S. in Physics and a B.S. and M.S. in electrical engineering from Portland State University. He has been developing embedded systems and medical devices for over 10 years, and is currently the project manager for the Portland State Aerospace Society, an educational aerospace project developing open source and open hardware sounding rocket systems.

Takuji Ebinuma obtained his B.S. and M.S. in Computer Science from the Kyushu Institute of Technology, Japan, and his PhD in Aerospace Engineering from the University of Texas at Austin. Since April of 2005 he has been working on various GNSS projects at the Mitsubishi Electric Corporation in Japan.

ABSTRACT

Open source software running on inexpensive, commercially available GPS receivers promises to cut the expense and restrictions of developing extended GPS applications dramatically. Extended GPS applications, such as keeping GPS satellite signal lock in high-dynamic environments or deep coupling of inertial data, require direct access to the receiver's hardware. This access is usually only provided by very expensive and restrictive development kits, many of which still do not grant the developer direct access to the receiver chip set. Our software allows GPS developers to convert an inexpensive, commercially available GPS receiver into an open source development system that provides complete access to the receiver's hardware. The software, licensed under the GNU General Public License (GPL), is called GPL-GPS.

This paper describes the motivations, principles and technical development of the GPL-GPS project. This includes choosing a GPS receiver chipset, choosing a receiver board, and porting the software necessary to make a fully functional open source GPS receiver and receiver development system. The paper concludes with results from a static positioning test, results from a GPS simulator running a simulated sounding rocket trajectory, and future work.

GPL-GPS currently supports the Zarlink GP4020, a 12-channel L1 C/A GPS receiver baseband processor with an integrated 32 bit ARM7TDMI microprocessor, because of its open and extensive documentation. Commercially available receivers, coupled with an open hardware carrier board, can create a full featured development kit for less than US$300.

GPL-GPS is intended to be part of an inexpensive 6 degree-of-freedom differential GPS-aided inertial navigation system designed for a small, inexpensive, and open source sounding rocket platform being developed by the Portland State Aerospace Society.

MOTIVATIONS AND USES

Most applications requiring position, velocity and time (PVT) information can use inexpensive commercial "off-the-shelf" (COTS) GPS receivers to obtain this information. COTS receivers usually provide update rates around 1 Hz, operate only under low receiver dynamics, and can, in general, only be customized in trivial ways (such as changing the receiver's communication protocols).

Much more expensive COTS receivers can handle specialized, or extended, GPS applications. For example, specialized COTS receivers exist for high dynamic vehicles (launch vehicles and satellites), timing transfer, and precise positioning. However, often these specialized COTS receivers are too expensive for a given application and are not configurable enough to meet the needs of particularly specialized applications. For example, the need to customize the receiver is often necessary for:

Sensor integration: Novel automotive dead-reckoning systems and custom inertial measurement units for unmanned aerial vehicles (UAVs) often demand a level of integration that exceeds what is available commercially.

Pseudolite use: The use of pseudolites with custom PRN numbers also requires direct access to the receiver hardware.

Deeply embedded receivers: Size, weight and power consumption can be reduced by using the spare resources of the receiver's processor to run custom software along with the standard receiver code. This can eliminate the need for a separate application-specific processor.

Research: Most research on tracking loops, GPS signal propagation, and sensor integration require direct access to the receiver's configuration.

A major part of our motivation in developing GPL-GPS has been to address problems encountered while developing small sounding rockets for the Portland State Aerospace Society.

The Portland State Aerospace Society (PSAS) is an educational research project at Portland State University that is developing inexpensive avionics for small sub-orbital launch vehicles [1]. PSAS wanted a GPS receiver that could operate in the high dynamic environment of a sounding rocket and one that could eventually be tightly coupled to an IMU.

Many COTS GPS receivers are hardware-capable of high dynamic and tightly coupled operation, but to realize these capabilities a developer must be able to customize the software running on the receiver.

As embedded systems software and hardware designers, we were very surprised at the barriers GPS developers face. While most silicon chipset manufactures publish detailed chip specifications to the web, freely distribute samples of their chips, sell inexpensive development boards and even provide useful free software, GPS chipset manufacturers in general do not. Most currently require an extremely expensive software development kit (which may be tens of thousands of dollars and still only include a binary API to the chipset) and a restrictive non-disclosure agreement (NDA) that prevents the developer from sharing any source code written for the chipset. This inability to share code has undoubtedly forced many GPS developers to "reinvent the wheel"; that is, to create a basic software and algorithmic infrastructure that others have already written.

Developers usually assume that the chipset manufacturer is there to help, not hinder their use of a chipset. Clearly, a more streamlined and open GPS development environment, in line with an embedded systems development model, is needed.

While searching for such a model, we discovered Dr. Clifford Kelley's work on the OpenSource GPS (OSGPS) project. Kelley took an inexpensive commercial GPS receiver, removed its microprocessor, and connected the correlator chip directly to a 486 PC using an ISA prototyping board [2]. Although his OSGPS receiver system couldn't easily be used in a power, space, and weight constrained environment, it did provide a code base from which to start an open source COTS receiver project.

Using OSGPS as a starting point, we began our work in early 2003. By November of 2004, Ebinuma had completed his ARMGPS project and had the first fix of an open source COTS receiver. ARMGPS was then merged with GPL-GPS, and the new receiver code obtained a first fix on May of 2005.

By encompassing open source methodologies, tool suites, operating systems, and open hardware designs, the GPL-GPS project has become more than "just" open source software for GPS receivers: it is now an open, affordable and complete GPS receiver development system.

DESIGN PRINCIPLES

GPL-GPS is based on the "open source" design philosophy: the more open a system is, the more dynamic and vibrant it becomes as it is adopted and extended by multiple users with different needs. Linux is a prime example of such an open system; thousands of people around the world have moved Linux from a small educational project to a modern, full-featured desktop and enterprise-level operating system. GPL-GPS has six guiding design principles based on the open source design philosophy:

Open Source: All software must be open source. This includes application code, operating system, and software development tools. Using proprietary code or tools would be expensive, overly restrict end users, and require tedious and potentially litigious management of intellectual property.

Open Hardware: The required hardware (the GPS chipset and receiver board) must have open and available documentation to avoid having to reverse engineer and/or "hack" hardware.

Meant for deeply embedded applications: the receiver should be usable in deeply embedded applications like sounding rockets and UAVs. This means attempting to minimize size, weight, and power requirements. It also means that the receiver should not require an external processor, although it should be compatible with external processors when available.

Portable software: GPL-GPS code should be as cross-platform as is reasonable, maximizing its use in both future GPL-GPS receivers and in other open source projects such as OSGPS. This means minimizing the dependence of the software on the details of a

specific hardware platform. Also, the host development system should run on as many operating systems (e.g., Linux, Windows, Macintosh) as possible.

Inexpensive: The hardware cost of the system must be as low as possible (roughly a few hundred dollars) and be widely available.

Available: All software and documentation should be made available on the Internet when possible. Any developer with modest experience who is willing to read the documentation should be able to quickly and easily get the system up and running.

MAKING GPL-GPS OPEN SOURCE

There are many open source software licenses available. The GNU General Public License, or GPL, was chosen for this project because of its general acceptance as the standard open source license.

The GPL states that any *distribution* of a licensed program (in any form) must make the source code available. The requirement to make the source code available also applies to programs that are based on the original program, even those with extensive modifications. Note that this requirement does *not* include programs that simply interact with the open source code.

In essence, the requirement to make the source code available when distributing the program makes it impossible to have proprietary source code that is based on open source code. While some see this as restrictive, the free software community views this more as a requirement for participation than a restriction. People who benefit from open source should expect that their modifications to that code should be open as well (again, if they choose to distribute them) [3].

SELECTING A RECEIVER CHIPSET AND BOARD

A GPS chipset is commonly thought of as the integrated circuits (ICs) necessary to decode and process GPS signals. Most GPS receivers require one to three ICs: a radio frequency (RF) front end (a downconverter and analog-to-digital converter), a baseband processor (or correlator), and some kind of general-purpose microprocessor. In practice, commercial GPS chipsets come in five configurations:

Front-end Only: Software-defined radio (SDR) receivers use just a RF front end IC (a downconverter and ADC) to receive GNSS signals. These receivers process the signals directly in software, with no other hardware support.

Correlator only: A single custom chip (an application specific integrated circuit, or ASIC) that only does GPS correlation. It requires a separate RF front end and microprocessor.

Combined correlator/processor: A single chip that contains both the correlator and the microprocessor. It requires only a RF front end.

Single chip receiver: A single chip receiver that combines a RF front end, a correlator, and a processor.

Soft Correlators (FPGA): A correlator implemented in a configurable Field Programmable Gate Array (FPGA) instead of in a custom designed ASIC.

It can be argued that an open source receiver should move away from a hardware-dependent GPS chipset toward a SDR system. This would increase the flexibility of the receiver while reducing the cost and complexity of the receiver hardware. Unfortunately, moving from a GPS correlator chipset to a general purpose processor or FPGA currently incurs unacceptable power and size penalties for the embedded applications that GPL-GPS is designed for: ASIC-based receivers generally require 9 - 36 cm^2 and consume less than 1 W of power, while general purpose processors and FPGAs require at least 50 - 150 cm^2 and consume 10 - 50 W of power.

As a stand-alone receiver, the chipset for GPL-GPS requires:

- Open and accessible chipset documentation (it cannot require a non-disclosure agreement).
- A microprocessor architecture which does not require an extraordinary amount of effort in either software or hardware development, and is supported by open and available tools.
- An inexpensive and available COTS receiver board.

There are currently about a dozen commercial GPS chipsets on the market. Of these, many are not applicable to GPL-GPS: many were not user programmable as a stand-alone receiver, others were "position peripheral" chipsets meant for deeply embedded consumer applications such as cellular phones.

Of the many applicable chipsets, only one had open and available documentation. This means *every* other chipset required some kind of license agreement with the chipset manufacturer, all of which included a non-disclosure agreement.

The only open and available chipset at the start of the GPL-GPS project were from Zarlink Semiconductor, which continues to produce chipsets based on GEC Plessey's GP1010 and Mitel's GP2021 chipsets. There are two Zarlink chipsets available: the GP2021 baseband correlator, which is simply a GPS ASIC correlator, and the GP4020 Baseband Processor, which is the same ASIC correlator as the GP2021 but is integrated with

ARM7TDMI microprocessor. All architectural and application notes are available on Zarlink's web site, including reference designs for receiver boards.

Since no GP2021-based stand-alone COTS receiver boards are available, *only one chipset* was usable by the GPL-GPS project: Zarlink's GP4020.

Fortunately, using the GP4020 had several important advantages. Because the GP4020 and GP2021 share essentially the same GPS correlator design, Kelley's OpenSource GPS receiver project could be easily ported from one to the other. Also, the GP4020's microprocessor is based on the ARM7TDMI, a widely used 32 bit RISC architecture with available open source tools.

A typical GP4020-based receiver is shown in Figure 1. Note the antenna, the Zarlink GP2015 L1 RF front end, and the 2 bit ADC output (sign, magnitude) that is fed into GP4020 correlator block.

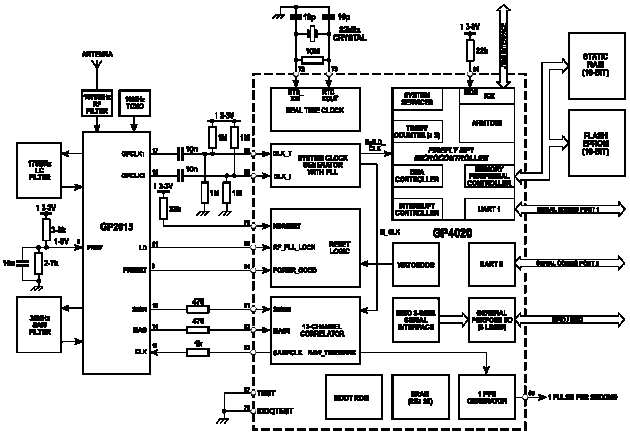

Figure 1: GP4020-based receiver block diagram (from GP4020 GPS Baseband Processor Design Manual [4]).

With the chipset selected, we then chose a specific Original Equipment Manufacturer (OEM) GPS receiver board (although our code should run on almost any GP4020-based board). The OEM board requires a receiver that:

- Uses the Zarlink GP4020 Baseband Processor chip.
- Has enough RAM and ROM (> 128 kB).
- Does not need to have its hardware modified in any way.

This last requirement is important because relying on modifying the hardware on a receiver board severely diminishes the number of software-only developers willing to work on a project. Other important considerations for the receiver are:

- Access to the GP4020's pins, in particular the pin used for the GP4020's built-in boot loader and the general-purpose I/O (GPIO) pins.
- Compact size and on-board power management.
- Standard RF connector (e.g., MCX)
- Widely available.
- Relatively inexpensive.

See Table 1 for a comparison of the five commercially available boards based on the GP4020. Despite its higher cost, the SigNav MG5001 is the clear choice as a development receiver for two reasons:

- Larger RAM size (useful for rapid development)
- The high density 51 pin connector allows the MG5001 to be mounted as a daughter board on a development board, providing direct access to the GP4020's serial ports, JTAG debugging pins, serial boot pin, GPIO pins, data lines and a small subset of address lines.

SigNav's other GP4020-based receiver, the MG5003, has the advantage that it has an extremely small form factor and has a large amount of RAM. But the lack of a large connector make it difficult to access to the GPIO lines, and the use of serial flash forces the software to run completely in RAM, partly negating the advantage of the larger RAM size.

The NovAtel SuperStar II™ is half the cost of the MG5001, but the added expense of the MG5001 seems worth the added development flexibility of the 51 pin connector and the larger memory. Not surprisingly, however, the SigNav MG5001 and the NovAtel SuperStar II are very similar to each other, and there are almost no differences between them from a software development point of view. Further, since only development activities require a large RAM size, the SuperStar II is a good choice for GPL-GPS receivers being installed into an application after software development has been finished.

Table 1
Zarlink GP4020-based COTS GPS Receiver Boards

Mfg.	SigNav	SigNav	NovAtel	Surrey Sat. Tech. Ltd (SSTL)	German Space Agency (DLR)
Model	MG5001	MG5003	SuperStar II™	SGR-05	Phoenix
RF FE	GP2015	GP2015	GP2015	GP2015	GP2015
SRAM (kB)	256	512	128	256	256
Flash (kB)	256	256 (serial)	256	256	256
Cost (US)	~$250	~$250	~$125	NA	NA
Notes	51pin connector	Small form factor	32 kB EEPROM	MG5001-based space receiver	MG5001-based space receiver

Finally, DLR's Phoenix and SSTL's SGR-05 receivers are based on the SigNav MG5001. Both the Phoenix and SGR-05 are meant to be COTS space borne receivers and are both in preliminary trials. We expect them to be substantially more expensive than the MG5001 given their intended use.

GPL-GPS DEVELOPMENT BOARD

Since the SigNav MG5001 GPS receiver is meant to be a OEM receiver board, it needs a development or "carrier" board in order to access its various signals. We designed a 9.5 x 11.5 cm carrier board for the MG5001 (see Figure 2) that includes two DB-9 connectors with a 5 V to RS-232C signal level converter chip, switches for various GP4020 modes (including for the serial boot loader and JTAG debugger), 0.1 inch headers for access to the 51 pin connector's signals, power supplies, LEDs, and a small prototyping space.

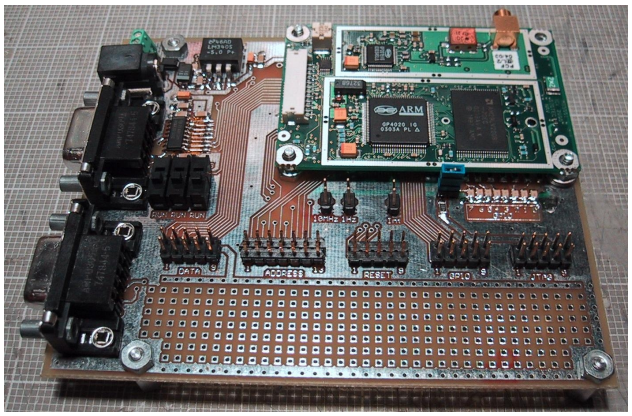

Figure 2: Open hardware GPL-GPS carrier board

The schematic and board layout was created in EAGLE CAD, a freely available (but not open source) PCB CAD development tool [6]. The design is posted on the GPL-GPS web site for others to download and manufacture [5]. A commercial circuit board manufacturing run costs roughly US$75 for two carrier boards, and there is approximately US$25 worth of components on the board.

SELECTING AND PORTING A REAL-TIME OPERATING SYSTEM

Although hard to quantify because of the lack of open documentation, it appears that most COTS GPS receivers rely on either custom real-time operating systems or simple run-time environments. In other words, the entire code infrastructure for the receiver was developed "in-house" for a specific receiver. While this can improve performance and reduce memory requirements, it is also a recipe for disaster when it comes to maintainability, portability, and enabling parallel development by multiple users. Even more important to an engineering project is enabling developers to focus on the *application* software rather than on the software infrastructure. For all of these reasons, we chose to use an already existing, standards-based, open source software infrastructure -- an operating system -- rather than forcing developers to learn a custom, bare bones and most likely buggy infrastructure.

The GPL-GPS project has very strict operating system requirements because of the high performance and low memory constraints. The requirements are:

Hard real time performance: Interrupt latencies must be on the order of 10 μs in order to handle the 1 ms accumulator interrupts, and context switches must be fast and deterministic in order to enable the use of threads. That is, the operating system needs to be a real-time operating system (RTOS) with guaranteed interrupt and task-switching latencies.

Small memory footprint: With only 256 kB of flash memory, the RTOS may take up no more than 128 kB (based on the OSGPS v1.17 current code size of 180 kB for the x86 processor).

Open source: As outlined in the GPL-GPS design principles, the RTOS must be open source with no licensing restrictions.

Existing port to ARM7 processors: Porting a RTOS to a microprocessor is out of the scope of a GPS receiver project; the RTOS must already be ported to ARM7TDMI processors.

Integrated debugging tools: Built-in debugging features are one of the most overlooked aspects of an

Table 2
Top Six RTOS Choices

RTOS	Custom	µClinux	Nucleus™	ISOS	µC/OS-II™	eCos
Hard real time	Y	N	Y	Y	Y	Y
< 128 kB footprint	Y	N	Y	Y	Y	Y
Open source	Y	Y	N	Y	N (published)	Y
Supports ARM7TDMI	Y	Y	Y	Y	Y	Y
Built-in debugging	N	Y	Y	N	N	Y
Rich IPC features	N	Y	Y	N	Y	Y
Simple configuration	N	Y	Y	Y	Y	N
Supports most proc.	N	Y	Y	N	Y	Y
Existing documentation	N	Y	Y	N	Y	Y

operating system. In order to make cross-platform embedded development tolerable, the RTOS must support remote loading *and* debugging of application software on the target system.

Some important considerations for choosing a RTOS include:

- **Rich operating system features** (abstractions like threads and interprocess communication primitives (such as mutexes, semaphores, etc) provide tools to dramatically simplify the application)
- **Simple to compile and configure**
- **Supports multiple architectures**
- **Good documentation**

A comparison of some of the operating systems considered for GPL-GPS is shown in Table 2.

The eCos real time operating system turned out to be the only applicable RTOS for this project. Fortunately -- like the chipset selection - it happens to be an extraordinary good fit to the requirements. eCos is:

- A hard real time operating system with low interrupt latencies (approximately 8 µs for an ARM7TDMI at 20 MHz),
- Integrated with "GDB stubs", a powerful and open source remote debugging tool,
- Tiny enough to fit in under 80 kB of memory,
- Free and open source (under the GPL),
- Ported to more than 10 different processor architectures (including x86, PowerPC, and SPARC)
- Well documented via an online reference manual and a published book,
- Uses the GNU software development toolchain, which includes free and open source tools such as the gcc compiler and gdb debugger [7].

The main drawback of using eCos is that it is an extremely complicated RTOS with an incredibly complex configuration system. Fortunately, users not interested in changing the configuration can use an already configured version of eCos distributed through the GPL-GPS project.

Note that eCos is not a "standard" operating system in the sense that it runs independently on the target system. Instead it is a "runtime library", a static pre-compiled program that the application links against at compile time. The library provides all of the functions of a standard operating system: startup, RTOS kernel, scheduler, etc.

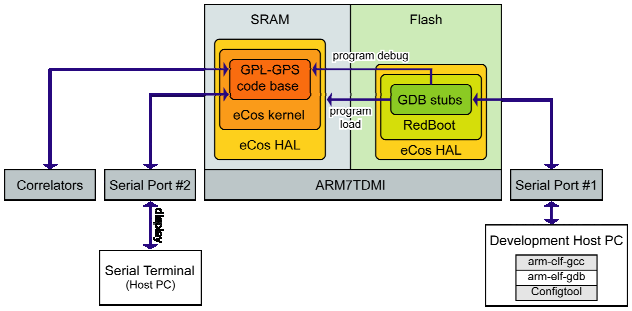

Figure 3: eCos software infrastructure

Figure 3 shows the GPL-GPS software infrastructure on a GP4020-based receiver. eCos is first compiled as a stripped-down standalone application, called RedBoot, and installed into the flash memory of the receiver. RedBoot acts as a boot loader, a small program which helps load applications into RAM. RedBoot includes such services as a simple command line interface, serial communication with up and download protocols, flash drivers, and debugging executives like GDB stubs. The host development PC communicates with RedBoot over a standard serial port to load and debug application programs in the receiver's RAM.

Once RedBoot has been installed in flash memory, the receiver is ready for GPL-GPS application development. The host PC compiles and links the GPL-GPS application with the eCos library, and downloads it using GDB stubs running in the RedBoot flash image. The host PC can now run and debug the program, using the GP4020's second serial port as a serial terminal to display application information.

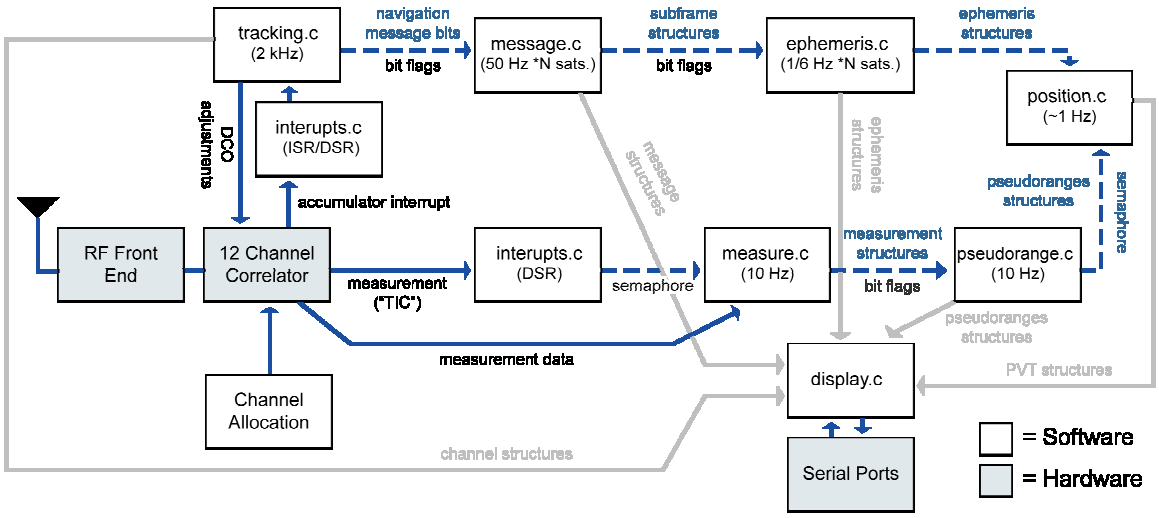

Figure 4: GPL-GPS task and data flowchart

Detailed instruction on installing RedBoot and the GPL-GPS development system can be found on the GPL-GPS website [5].

Finally, once the developer is satisfied with the receiver's performance, the software can be recompiled and written directly to flash. At this point, the receiver will automatically begin running the software on power-up and is ready for use in an embedded environment.

PORTING OPENSOURCE GPS

OSGPS runs on an x86 PC with a Zarlink GP2021 correlator on an ISA or PCI card. It runs under DOS and uses a simple foreground/background (interrupt with mainline) architecture. All time dependent functions --- accumulator dump processing, satellite navigation message decoding, and measurement processing --- are called directly from a single timer interrupt. Navigation algorithms and display code run in the mainline task with communication between tasks accomplished using global variables [2].

Over a period of 8 months we took OSGPS v1.17, stripped it down, and began porting it to eCos running on a SigNav MG5001 receiver. Although the developmental nature of our project never makes it "truly finished", GPL-GPS now exists as a code-complete development environment for creating extended GPS applications with COTS receivers.

Of the roughly 6,300 lines of code in OSGPS, over 80% have been re-factored or removed in the transition to GPL-GPS. Part of the reason for this change is the dramatic difference in platform: OSGPS is meant to run on a 100MHz 486 or greater PC with over 640kB RAM and a file system, while GPL-GPS runs on a 20 MHz ARM7TDMI with 256 kB RAM and 256kB flash. Worse, typical throughput for GP4020 receivers is roughly 5 MIPS from flash memory and 10 MIPS from external SRAM. Only a small internal 8K SRAM cache allows the processor to reach 20 MIPS. This means that performance is much more of a concern in GPL-GPS than in OSGPS.

Another factor in the large code change is that OSGPS is based on a single threaded DOS environment. Using eCos allows GPL-GPS to break its code into threads, which allows tasks to be more clearly (and in some cases more efficiently) broken up, with clear lines of communications between them.

The GPL-GPS code base attempts to implement the following performance improvements and code enhancements:

GPS task-based threads: Possibly the most important optimization has been repartitioning the OSGPS code to reflect the various GPS receiver tasks (see Figure 4). Each task has been modularized into its own code and header file in order to clarify and prioritize the tasks. Each task has one thread and one data structure associated with it, making it a more modular interface for task additions and/or inserting interfaces to external communication links (for example, an external flight computer on a UAV).

Event driven processing: Instead of timer delays, GPL-GPS uses event-driven threads and interprocess communication (IPC), including flags, semaphores and mutexes.

Streamlined critical tasks: Tasks that didn't need to be in higher priority, higher rate threads (or interrupts) have been moved to lower priority, lower rate threads. For example, channel allocation and

navigation message processing were moved from the tracking loops to their own low priority threads.

Prioritized threads: Instead of giving all the tasks the same priority, the threads are partitioned and carefully prioritized in order to let time critical tasks execute before lower priority tasks.

GP4020-friendly variable types: Since the correlator uses 16 bit integers, much of the integer processing can be moved from 32 bit to 16 bit words to speed up access to the external 16 bit SRAM.

Fixed point arithmetic: GPL-GPS is slowly moving from double precision floating point to 32 bit fixed point integers in critical code (e.g., the tracking loops). Note that in low priority, infrequently run tasks (such as the positioning task), floating point is left in place for coding ease and clarity.

CURRENT STATUS

Kelley's OSGPS has the features you would expect from GPS receiver software: search and tracking of satellites (using a simple FLL for search and acquisition, and a simple 2nd order PLL for lock), navigation message decoding, almanac storage and processing, pseudorange calculation, position, navigation and time calculations, atmospheric corrections, carrier phase processing, and as of v1.18 an 8-state Kalman filter.

GPL-GPS removed many of these features to simplify the complicated porting process: it currently does not support almanac storage and processing, atmospheric corrections, carrier phase, and Kalman filtering. However, now that the software infrastructure is in place, rapid progress can be made re-implementing and extending these features.

Because of the lack of smoothing and atmospheric corrections, GPL-GPS positioning errors tend to be large and somewhat biased. Figure 5 is a scatter plot of 2,000 points taken from a static location. To confound the error, the static test location also has severe multi-path problems due to local features. Figure 6 is the same data from an isometric perspective.

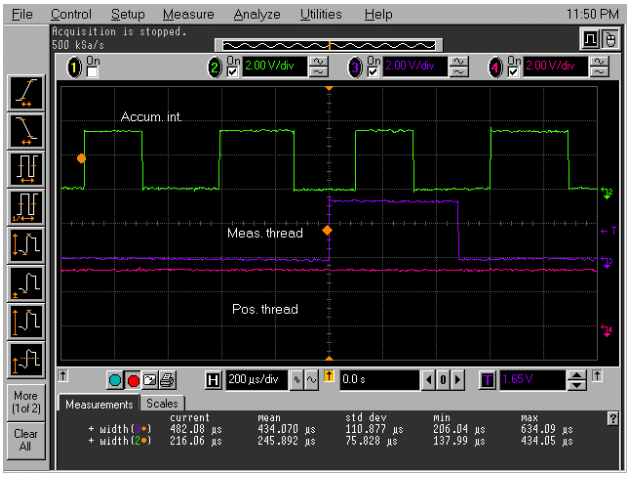

Figure 7**: GPL-GPS task timing**

Figure 7 shows the timing characteristics of three time-critical GPL-GPS tasks: the tracking loops, which run every 505 µs after an accumulator interrupt, and the measurement thread, which runs every 99.9999 ms after a measurement interrupt. The mean time spent in the tracking loops is 355 µs, or about 70% of the processor's time. This underscores the need to optimize the tracking loops by moving them into the GP4020's fast internal SRAM and changing them to use fixed-point math. The measurement thread takes 380 µs (note that it is interrupted and suspended while the tracking loop runs). The position thread takes between 400 ms (for four

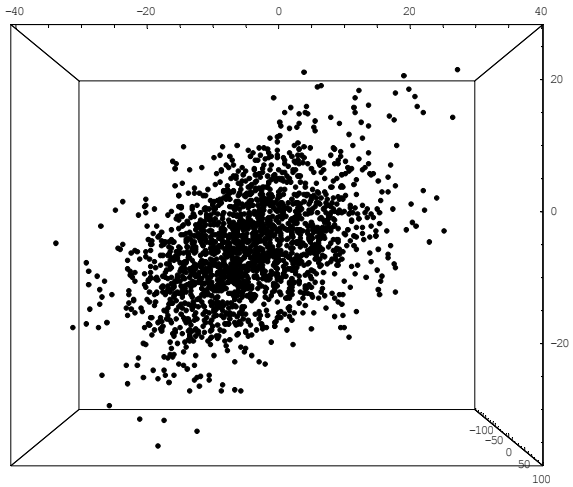

Figure 5: 2,000 points taken from a static position. Scale in meters along ECEF axes.

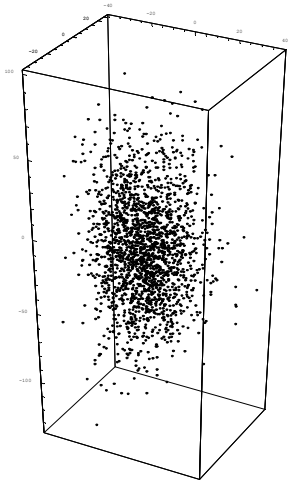

Figure 6: Same 2,000 points from an isometric perspective.

satellites) and 600 ms (for seven satellites).

Because GPL-GPS is meant for high dynamic vehicles, such as the small-scale sounding rockets being developed by PSAS, we have used more robust tracking loops than the original OSGPS. The receiver starts pre-flight with a standard FLL/PLL acquire/track until navigation messages are acquired. Then it switches to a full FLL tracking loop for flight to keep lock in a high dynamic environment [8].

Figure 8 shows a simulated small sounding rocket flight to 8 km with peak accelerations of 12 g's and a peak velocity of 550 m/s.

This trajectory was entered into a Spirent STR4760 12-channel L1 C/A code GPS simulator [9] and used to simulate a sounding rocket flight. Note that ionospheric and tropospheric delays were disabled in this simulation.

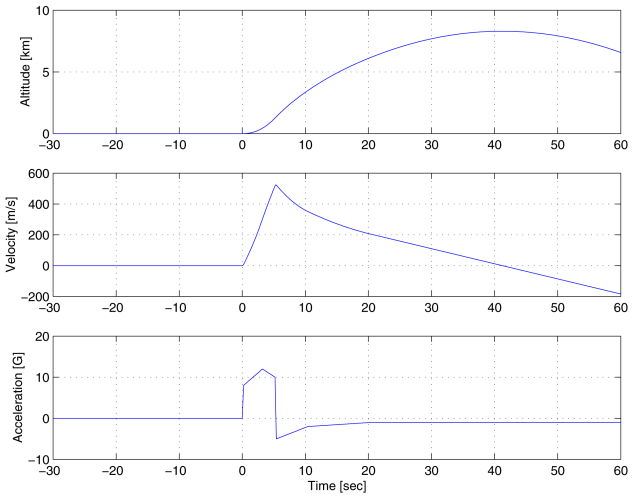

Figure 8: Simulated sounding rocket dynamics

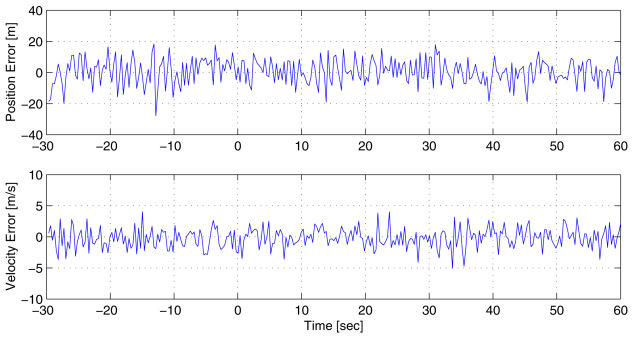

Figure 9: GPL-GPS position and velocity errors to a simulated sounding rocket flight.

Figure 9 shows the GPL-GPS positioning and velocity errors to the simulated sounding rocket flight. The receiver stayed locked throughout the flight, and had a position error of 8.1 m RMS and a velocity error of 4.0 m/s RMS. Maximum error during the simulation was 27.9 m and 5.0 m/s.

FUTURE WORK

The following work is currently in progress:

- The eCos port to the GP4020 is being optimized to support faster context switches and use the fast internal 32 bit SRAM of the GP4020 for the tracking loops.
- OSGPS features are being re-implemented, such as almanac processing and storage, atmospheric corrections, and carrier phase processing.
- GPL-GPS code that extended OSGPS features will be re-integrating with the originating OSGPS project.
- A network application programmer's interface (API) is being added at multiple levels to push information off of the GP4020 to an external microprocessor. For example, this enables GPL-GPS to rapidly compute and send pseudoranges to an external microprocessor (flight computer) that will then do more sophisticated (and thus computationally intensive) PVT calculations.
- A differential GPS code base is being developed to enable GPL-GPS to generate differential corrections and use them in its position calculations.

CONCLUSION

By porting open source software to commercial, off-the-shelf GPS receivers, GPL-GPS lowers the barriers to GPS receiver development. Its inexpensive and open nature makes possible a rich variety of new and interesting GPS applications. Although still in its infancy, GPL-GPS is already being considered for projects ranging from nano-satellites, UAVs, and small-scale launch vehicles to low cost DGPS navigation systems and yard maintenance robots. The hope is that these varied and interesting GPL-GPS users will then contribute their code improvements back to the project, enabling GPL-GPS to quickly grow in maturity and features.

For more up-to-date information on the status of the GPL-GPS project (as well as other open source and open hardware GNSS projects) please see http://gps.psas.pdx.edu/

ACKNOWLEDGMENTS

The authors would like to acknowledge Dr. Clifford Kelley for his pioneering work on OpenSource GPS, and Tim Brandon, Jamey Sharp, Frank Mathews and Gary Thomas for their work on the GPL-GPS project.

REFERENCES

[1] The Portland State Aerospace Society. PSAS homepage. http://psas.pdx.edu/

[2] Kelley,C., Barnes, J., Jingrong, C. OpenSource GPS: Open Source Software for Learning about GPS. In *15th Int. Tech. Meeting of the Satellite Division of the U.S. Inst. of Navigation*. Institute of Navigation, September 2002.

[3] The Free Software Foundation. General Public License. http://www.gnu.org/licenses/gpl.html, September 2005.

[4] Zarlink Semiconductor. *GP4020 GPS Baseband Processor Design Manual (DM5280)*, January 2002.

[5] The GPL-GPS Project. http://gps.psas.pdx.edu/ September 2005.

[6] CadSoft Software. EAGLE CAD Homepage. http://www.cadsoft.de/, September 2005.

[7] The eCos real Time operating system. eCos Homepage and on-line documentation. http://ecos.sourceware.org/, September 2005

[8] Bo Son, S., Kyu Kim, I., Heon Oh, S., Hwan Kim, S., Baek Kim, Y. Commercial GPS Receiver Design for High Dynamic Launching Vehicles. In *2004 International Symposium on GNSS/GPS*, GNSS Sydney 2004, December 2004.

[9] Spirent, PLC. Spirent homepage. http://www.spirent.com/, September 2005