

2005

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

Andrew Greenberg
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation


Greenberg, Andrew, "Open Source Software for Commercial Off-The-Shelf GPS Receivers" (2005).
Dissertations and Theses. Paper 6603.


This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.


THESIS APPROVAL

The abstract and thesis of Andrew Greenberg for the Master of Science in Electrical and Computer Engineering were presented May 6, 2005, and accepted by the thesis committee and the department.


COMMITTEE APPROVALS:


James McNames, Chair


Y.C. Jenq


Bart Massey
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:


Malgorzata Chrzanowska-Jeske, Chair
Department of Electrical and Computer
Engineering

OPEN SOURCE SOFTWARE FOR COMMERCIAL
OFF-THE-SHELF GPS RECEIVERS

by
ANDREW GREENBERG

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
2005

ABSTRACT

An abstract of the thesis of Andrew Greenberg for the Master of Science in Electrical and Computer Engineering presented May 6, 2005.

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

Inexpensive and commercially available Global Positioning System (GPS) receivers can be used in novel applications, such as nanosatellites and unmanned aerial vehicles (UAVs), by modifying the receiver's software. Unfortunately, typical GPS software development systems cost tens of thousands of dollars and have restrictive license agreements, such as non-disclosure agreements, for both their hardware and software. This thesis describes the design and construction of an open source GPS receiver development system that addresses these unnecessary restrictions. For the first time, any developer with a PC, a commercially available GPS receiver, and this development system can quickly and easily develop software for extended GPS applications. The software is licensed as open source under the GNU General Public License (GPL) version 2 and is called GPL-GPS.

GPL-GPS currently supports Zarlink's GP4020 baseband processor, a 12 channel GPS correlator with an integrated 32bit ARM7TDMI microprocessor. Any GP4020-based receiver with 256 kB or more of RAM and flash memory is able to run GPL-GPS. When coupled with a commercially-available GP4020-based receiver and an open hardware development board, GPL-GPS provides a full-featured GPS experimentation and application development kit for less than US \$200.

Dedicated to

my lovely and talented wife

Jennifer.

Acknowledgements

Like many cross-discipline studies, this thesis could never have been completed without the help of people who know far more than the author.

In particular, I'd like to acknowledge four groups of people: Tim Brandon and Jamey Sharp for their enormous contributions to this thesis and my development as an engineer and programmer, Clifford Kelley and Takuji Ebinuma for their ground-breaking work on the OpenSource GPS software, and Gary Thomas for his authorship of and contributions to the eCos real time operating system. Finally, I would be remiss not to acknowledge the teeming hordes of open source programmers who have built a nice set of shoulders for the rest of us to stand on.

I am indebted to Professor James McNames, a great adviser with a long term vision. Professor Bart Massey encouraged me to get this done once and for all and helped me with L^AT_EX, and Morgan Odland was kind enough to translate my rough hand sketches into skillfully formatted figures.

Finally, thanks to the 'fam' for bearing with me for seven long years.

Thank you all.

Table of Contents

List of Tables	v
List of Figures	vi
1 Introduction to GPS	1
1.1 The Global Positioning System	1
1.1.1 Types of GPS Receivers	2
1.1.2 Radio Frequency Trilateration	2
1.1.3 Satellites: The Space Segment	4
1.1.4 GPS Signal Description	5
1.2 GPS Receivers: the User Segment	6
1.2.1 Antenna	6
1.2.2 Prefilter and LNA	7
1.2.3 RF Front End (Down-converter and ADC)	7
1.2.4 GPS Correlator	7
1.2.5 Extracting the Navigation Message	12
1.2.6 Taking Measurements	13
1.2.7 Calculating Pseudoranges	15
1.2.8 Calculating Position and Time	15
1.2.9 Carrier Phase Velocity and Position	16
1.2.10 Filtering Position Output	17

1.2.11	GPS Receiver Software	17
1.2.12	GPS Receiver Performance	18
1.3	GPS Receivers: Applications	18
1.3.1	Standard GPS Applications	19
1.3.2	Specialized GPS Applications	20
1.3.3	Extended GPS Applications	21
2	Motivation and Principles behind GPL-GPS	23
2.1	Motivations for an Open GPS Development System	23
2.1.1	The Embedded System Development Model	23
2.1.2	Debugging Critical Flight Hardware	25
2.2	Moving Towards Open Source Solutions	26
2.3	GPL-GPS Design Principles	28
2.4	Immediate Applications for GPL-GPS	29
3	Selecting a Receiver Chipset and Board	31
3.1	Choosing a GPS Receiver Chipset	31
3.1.1	GPL-GPS Chipset Requirements	32
3.1.2	Applicable GPS chipsets	33
3.1.3	Choosing a Chipset	36
3.2	Choosing a GP4020-based GPS Receiver	38
3.2.1	GP4020-based GPS Receiver Requirements	38
3.2.2	Available GP4020-based GPS Receivers	38
3.2.3	Choosing a Receiver	39
3.2.4	Creating a GPL-GPS Development Board	40

4	Selecting and Porting a RTOS	42
4.1	‘Soft’ vs. ‘Hard’ Real Time	42
4.2	RTOS Requirements	43
4.3	Applicable Real Time Operating Systems	44
4.4	Choosing a RTOS	46
4.5	eCos Architecture	47
4.6	Porting eCos to the GP4020	49
4.7	GPL-GPS Software Infrastructure	50
4.8	Getting eCos Running on the GP4020	51
5	Porting OpenSource GPS	53
5.1	Introduction to OpenSource GPS	53
5.2	ARMGPS: Porting OSGPS to the GP4020	55
5.3	Transforming ARMGPS to GPL-GPS	56
5.3.1	GPL-GPS Computing Environment	56
5.3.2	Optimizing for GPL-GPS	57
5.3.3	Improving on OSGPS Algorithms	59
5.4	GPL-GPS Current Status	60
5.5	GPL-GPS Current Performance	61
6	Future Work and Conclusion	63
6.1	Future Work	63
6.1.1	Introduction	63
6.1.2	Reimplement Atmospheric Correction	63
6.1.3	Reimplement Almanac Processing	64
6.1.4	Reimplement Carrier Phase Tracking	64

6.1.5	Faster Cold Acquisition	64
6.1.6	Move Tracking Loops to Internal SRAM	65
6.1.7	Code Refactoring and Contribution back to OSGPS	65
6.1.8	Better Phase Locked Loop Algorithm	66
6.1.9	Network API (Flight Computer Model)	66
6.1.10	Lock Aiding	66
6.1.11	DGPS	66
6.1.12	Attitude	67
6.1.13	Moving towards Open Hardware	67
6.2	Conclusion	67
Bibliography		69
Appendix A Initial Results		72
A.1	Introduction	72
A.2	SigTec OEM Software Positioning Results	73
A.3	GPL-GPS Software Positioning Results	74
A.4	Receiver Time Series	78
A.5	Comparison Conclusion	78
Appendix B The GPL-GPS Development Board		81

List of Tables

1.1	GPS error budget (from [15]).	19
3.1	Comparison of GPS chipsets.	35
3.2	Commercially available GP4020-based GPS receivers.	39
4.1	Comparison of real time operating systems.	45

List of Figures

1.1	The GPS constellation (reproduced from [18]).	5
1.2	Block diagram of the Zarlink GP2015 GPS front end (reproduced from [24]).	8
1.3	One dimensional (code phase only) Gold code correlation.	9
1.4	Two dimensional (code phase and carrier frequency) Gold code correlation: contour map.	9
1.5	Two dimensional (code phase and carrier frequency) Gold code correlation: 3D plot.	9
1.6	Block diagram of a channel in the Zarlink GP4020 (reproduced from [25]).	11
1.7	Tracking loops for a generic software receiver (reproduced from [21]).	12
1.8	Zarlink GP4020 correlator peripheral block diagram.	14
1.9	Generic GPS receiver tasks.	18
2.1	PSAS Launch Vehicle No. 1b (LV1b) GPS flight data from October 2000	26
3.1	Generic GP4020-based GPS receiver block diagram (reproduced from [25]).	37
4.1	The eCos <code>configtool</code> program configuring the <code>gps-4020</code> template.	48

4.2	GPL-GPS software architecture: GPL-GPS, eCos, RedBoot and host system.	51
5.1	The original OSGPS ISA PC card (reproduced from [9]).	53
5.2	OpenSource GPS software flowchart (reproduced from [9]).	54
5.3	GPL-GPS software flowchart.	59
5.4	GPL-GPS timing: tracking loops and measurement thread.	61
5.5	GPL-GPS timing: tracking loops, measurement thread, and position thread.	62
A.1	Comparison testing setup: roof-mounted antenna, antenna splitter and two MG5001 receivers on development boards	73
A.2	Initial ECEF positions of the SigTec OEM software	74
A.3	54,747 ECEF positions at 1 Hz from the SigTec OEM software	75
A.4	Histogram of range to the sample mean for the SigTec OEM software	75
A.5	GPL-GPS data bounded by a 1 km bounding box (98.8% of points)	76
A.6	GPL-GPS data bounded by a 100 m bounding box (90.5% of points)	77
A.7	Histogram of range to the sample mean for GPL-GPS (1 km bounded data set)	77
A.8	Time series graph from the SigTec OEM Software	78
A.9	Time series graph from GPL-GPS	79
A.10	Zoomed-in time series from GPL-GPS	80
B.1	Schematic of the GPL-GPS carrier board for the SigTec MG5001 receiver.	82

Chapter 1

Introduction to GPS

No matter where you go, there you are.

— *Buckaroo Banzai*

This chapter introduces the Global Positioning System (GPS) and the technical operation and applications of GPS receivers.

1.1 The Global Positioning System

The Global Positioning System, or GPS, is the world's first Global Navigation Satellite System (GNSS). A constellation of more than 24 satellites in medium earth orbit continuously broadcasts radio frequency signals which allow GPS receivers to determine their position, velocity, and time.

GPS receivers are becoming ubiquitous. Although best known as navigational aides for drivers and recreational hikers, GPS receivers are quickly becoming embedded in consumer technology such as cell phones, small electronic hand-held organizers, and, recently, wrist watches. As the integrated circuits for GPS receivers shrink in size and require fewer external components, we can expect GPS receivers to become increasingly embedded in our everyday experience.

1.1.1 Types of GPS Receivers

The term “GPS receiver” covers a wide range of hardware and software forms:

Consumer receivers are inexpensive (typically under US \$200) GPS receivers that include a display screen, input device (e.g., a keyboard), and an integrated microwave patch antenna.

GPS receiver boards are original equipment manufacturer (OEM) boards that are intended for applications requiring positioning or time data but not a direct user interface. Consumer receivers include GPS board functionality as well as user interface hardware.

Software receivers (often known as “software defined radios”) minimize hardware by using software to do the majority of signal processing.

In this thesis, the term “GPS receiver” will refer to a printed circuit board with all the electronic components necessary to receive and decode GPS signals but no user interface or extraneous hardware. GPS receivers have only a GPS chipset, a general purpose processor, memory (ROM and RAM), and the radio frequency components necessary to operate the receiver. These receivers are also known as “OEM receivers”, “GPS sensors”, and even “GPS engines”.

1.1.2 Radio Frequency Trilateration

Triangulation is the most common form of beacon-based position determination. Measuring the angles between a receiver and three beacons at known positions allows a receiver to determine its location. How accurately the receiver’s location

can be determined depends on how accurately the receiver can measure the angles involved.

GPS works by trilateration; knowing the distance to three beacons at known locations allows the receiver to determine its position. The accuracy of trilateration depends upon how accurately the receiver can determine the distance to the beacon. In the case of GPS, each satellite carries an atomic clock to synchronize the broadcast of a microwave signal. When the signal is received, the arrival time is recorded and compared to the time of transmission. The time between the transmission and reception of the broadcast is the “time of flight”: multiplying the time of flight by the propagation speed (in this case, the speed of light) provides the receiver with the distance to the beacon. Consequently, for a time-of-flight, trilateration-based positioning system, accuracy is limited by how accurately the receiver can measure time.

Measuring the time of flight requires that the receiver’s and the beacon’s clocks are precisely synchronized. Since radio frequency signals propagate at roughly the speed of light (about 0.3 m per nanosecond), meter-level positioning accuracy requires timing with nanosecond accuracy. GPS satellites and their ground-based control centers solve this problem with redundant atomic clocks. Unfortunately it’s not feasible to have atomic clocks in GPS receivers due to their expense and size. However, a GPS receiver’s inexpensive, temperature-dependent, frequency-drifting, and noisy crystal-based clock can be quickly and continuously synchronized with the navigation system’s time by adding time as a fourth unknown to the navigation problem. Solving this four dimensional (X, Y, Z, t) problem requires at least four independent measurements. With one measurement per satellite, a stand-alone GPS receiver must receive signals from four satellites, not three, to

synchronize its clock and acquire position.

If the receiver is given more information, fewer satellites are necessary to solve the position equation. For example, GPS receivers with altimeters can navigate with only three satellites since their position is now three dimensional (X,Y,t). And GPS receivers with very stable clocks can navigate with only three satellites until their clock drift gives unacceptable errors.

1.1.3 Satellites: The Space Segment

The core of GPS is a constellation of 24 satellites in medium earth orbit. Developed and deployed by the United States government, these GPS satellites are the “Space Segment” of the Global Positioning System.

The 24 satellites (or more, including in-orbit spares) are equally distributed in six longitudinally distributed orbital planes, each inclined 55 degrees to the equator at an altitude of 20,335 km (Figure 1.1). Depending on latitude, an average of 5 to 7 satellites are visible 5 degrees above the horizon at any given time [15]. Because the beamwidth of the GPS signal extends about 7.4 degrees past the limb of the earth, and because of unavoidable back propagation of the satellites’ antennas, GPS can also be used in low and medium earth orbit [2].

Each GPS satellite carries redundant Cesium and Rubidium atomic frequency standards (AFS) which steer a precision 10.23 MHz oscillator. The AFS, along with adjustments from the GPS ground controllers, keep the 10.23 MHz time base to within 6 ns of the GPS system time. This accuracy is required to keep systemic timing errors below 1.2 m. The satellites also use this precision time base to control the frequency of their broadcast signals: the GPS uses the L1 frequency of 1.57542 GHz, which is 154 times the 10.23 MHz timebase.

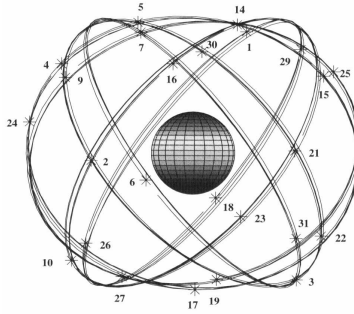


Figure 1.1: The GPS constellation (reproduced from [18]).

1.1.4 GPS Signal Description

GPS satellites use code division multiple access (CDMA) to share the L1 frequency between multiple satellites. Specifically, GPS uses Gold codes, a class of nearly orthogonal codes which can be used for CDMA transmissions due to their low cross-correlation. Each orbital satellite is assigned a pseudorandom index number (PRN) which denotes its Gold code and uniquely identifies the satellite. GPS satellites also have satellite vehicle designations, but are most often referred to by their PRN [15].

The Gold code sequence is periodic therefore each symbol conveys a negligible amount of information. This is why the Gold code symbols are called “chips”, rather than bits. Each chip of the 1,023 chip GPS Gold code sequence is binary phase shift keyed (BPSK) onto the L1 carrier at a chip rate of 1.023 MHz making the sequence repeat once each millisecond. Modulated on top of this 1.023 MHz signal is a 50 bit-per-second message that is also binary phase shift keyed onto the carrier. This 50 bps signal contains a 1,500 bit navigation message which includes the satellite’s health, clock corrections, a detailed description of its orbit

(ephemeris data), and a general orbital description of the entire GPS constellation (almanac data). Using the ephemeris data, the receiver can calculate where the satellite was (to < 0.5 m) at the time of transmission. Using the almanac data and the receiver's current position, the receiver can predict which satellites may be currently visible.

The satellite signals are broadcast at 50 W, but are attenuated by distance and the atmosphere to a guaranteed minimum level of a few microvolts (-130 dBm) at the Earth's surface. Since this is below the typical noise floor of roughly -90 dB, GPS signals are “under the noise floor” and can only be recovered by using noise rejecting techniques. For conventional GPS receivers, a local copy of the Gold code sequence is correlated with the received signal to recover the original signal while rejecting un-correlated noise [15].

1.2 GPS Receivers: the User Segment

A conventional GPS receiver, part of the GPS “user segment”, requires five components: an antenna, a radio frequency front end (RF front end), an analog to digital converter (ADC), a correlator with at least four channels (although in some cases this can be done in software), and a general purpose processor.

1.2.1 Antenna

Antennas for GPS receivers must acquire the right-hand circularly polarized signals from the satellites while minimizing multipath signals. Multipath signals are those signals which have been ‘bounced’ off of local features, increasing their propagation path length and thus changing their phase. These multipath signals

‘smear’ the phase of GPS signals, making precise phase tracking difficult and thus adding noise to the receiver’s position measurements.

1.2.2 Prefilter and LNA

The 1.57542 GHz signal from the antenna is typically prefiltered and amplified using a low noise amplifier (LNA) on the receiver board.

1.2.3 RF Front End (Down-converter and ADC)

Typically, GPS receivers have a “radio frequency front end” chip that down-converts and digitizes the 1.57542 GHz carrier into a signal a few megahertz wide. For example, the Zarlink GP2015 RF front end (Figure 1.2) uses a 10 MHz temperature-controlled crystal oscillator (TCXO) as the local oscillator for a three-stage down-converter. After down-conversion, the resulting 4.3 MHz signal is over-sampled by a two bit flash analog to digital converter at a sample rate of 5.7 MHz. This aliases the digitized signal, resulting in a final output frequency centered around 1.4 MHz. The 5.7 MHz digitized stream is sent to a correlator chip to be further processed.

1.2.4 GPS Correlator

Conventional GPS receivers are usually able to track 6–12 satellites. Each tracked satellite requires a ‘channel’, which is a set of correlation hardware that includes a Gold code generator, binary multipliers, and two digitally controlled oscillators (DCO)s, one for the carrier frequency generator and one for the Gold code generator.

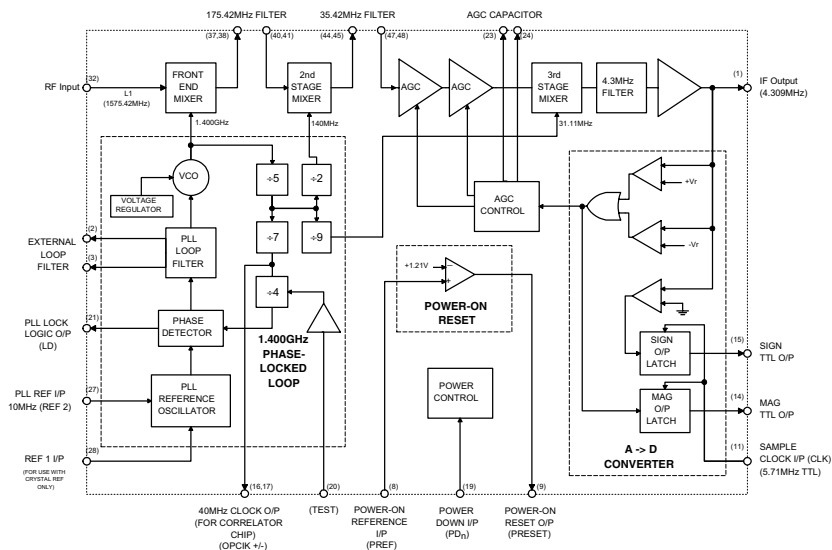


Figure 1.2: Block diagram of the Zarlink GP2015 GPS front end (reproduced from [24]).

The purpose of each channel is to replicate a satellite’s Gold code sequence and multiply it with the incoming signal. The summation of this product (or correlation) is very low if the code phase and frequency of the local replica doesn’t match the received signal’s phase and frequency. However, if the signals are synchronized, the correlation value spikes and the signal is detected. An example of a one-dimensional correlation in code phase only is shown in Figure 1.3, where the X axis is the relative phase of the two Gold code sequences.

The search for GPS signals — ‘acquiring’ the satellite signal — is a two dimensional search problem. First, the variable distance to the transmitting satellite causes an unknown phase shift in the incoming signal’s Gold code. Second, the satellite vehicle’s motion relative to the user causes an unknown Doppler shift in the carrier frequency. There are also second and third order effects on the code frequency due to dispersion of the signal in the ionosphere, but these smaller ef-

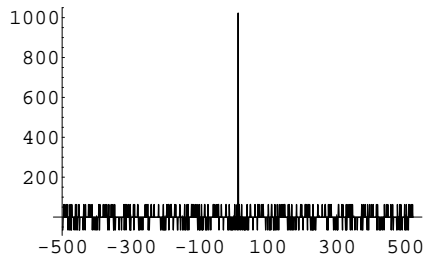


Figure 1.3: One dimensional (code phase only) Gold code correlation.

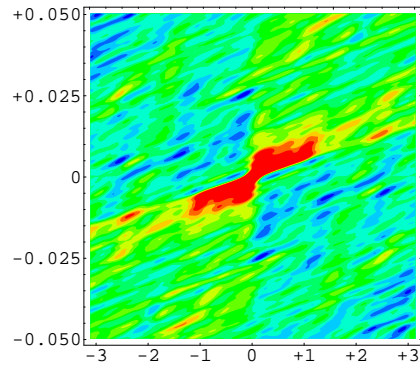


Figure 1.4: Two dimensional (code phase and carrier frequency) Gold code correlation: contour map.

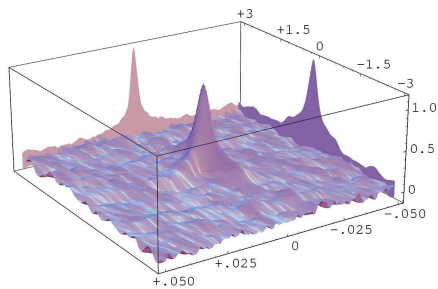


Figure 1.5: Two dimensional (code phase and carrier frequency) Gold code correlation: 3D plot.

fects can safely be ignored during acquisition. Figures 1.4 and 1.5 show full two dimensional correlation in code phase and carrier frequency.

For a practical example, a channel in the Zarlink GP4020 baseband correlator (Figure 1.6) uses a carrier frequency DCO to match the incoming carrier frequency and a code generator (run by a separate code frequency DCO) to match the incoming code phase and frequency. Each correlator channel has four accumulators, which are registers meant to hold the correlation's sum of products. Two of the registers accumulate the in-phase component (I) of the incoming signal, and two registers accumulate the quadrature component (Q) of the incoming signal. These accumulators are further divided into "prompt" and "tracking" sets. The prompt accumulators (I_{prompt} and Q_{prompt}) follow the incoming signal as closely as possible. The tracking accumulators ($I_{tracking}$ and $Q_{tracking}$), generate a signed error signal by subtracting an 'early' version of the code phase from a 'late' version of the code phase.

GPS receivers usually require a general purpose processor to control the channel's DCOs. This control loop is the processor's most time-intensive task, requiring typically 20% to 50% of the processor's bandwidth [21]. After every full Gold code cycle (every 1 ms), the processor must check the accumulators. If searching for satellites, the processor analyzes the values in the accumulators and decides whether to continue scanning code phase and carrier frequency for satellites, or to try and "pull in" a possible candidate signal. Once a satellite signal is found, the processor must run two control loops:

1. Monitor the error signal of the tracking accumulators every code cycle in order to control the code frequency DCO.

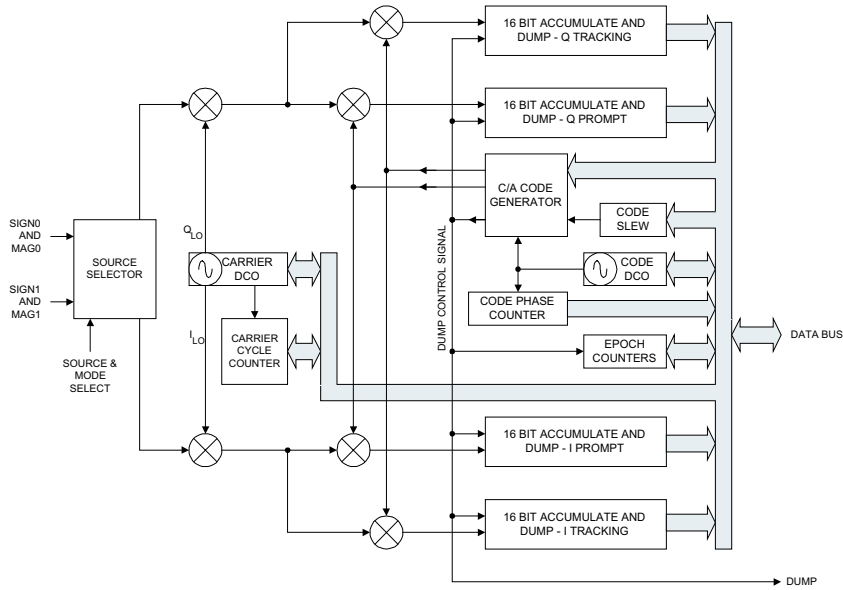


Figure 1.6: Block diagram of a channel in the Zarlink GP4020 (reproduced from [25]).

2. Maximize the value of the in-phase (I) prompt accumulator and minimize the quadrature phase (Q) prompt accumulator over some integration period in order to adjust the carrier frequency DCO.

Figure 1.7 shows a generic software tracking loop: toward the top of the figure, the code generator generates two Gold code sequences, one a half-chip early and one a half-chip late in phase. Subtracting the sum of products from these two phase-shifted codes produces an error signal which is averaged, summed and used to control the code generator. The resulting prompt, or in-phase code is multiplied against the input signal to remove the Gold code from the input to the carrier tracking loop. The carrier loop generates a phase-locked carrier signal by maximizing the in-phase (I) component and minimizing the quadrature (Q) phase of the signal. This carrier signal is then used to remove the carrier from the code

for the code loop.

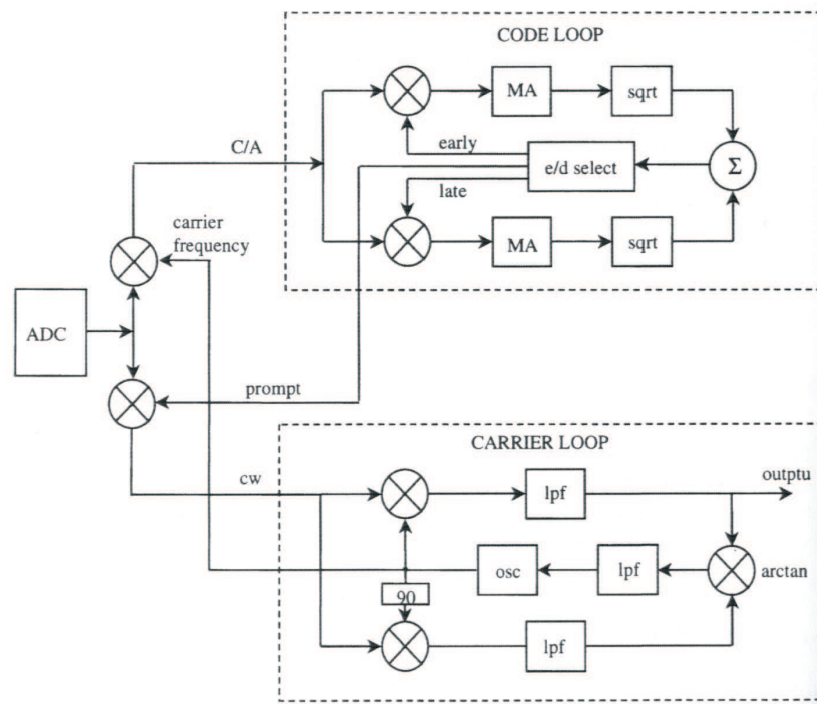


Figure 1.7: Tracking loops for a generic software receiver (reproduced from [21]).

The control of the DCOs can be done with a frequency-locked loop (FLL) during pull-in of a signal and a phase-locked loop (PLL) during lock. The FLL allows a wider capture range for acquiring the signal, but does not track the phase closely enough for the precision phase alignment necessary for accurate time-of-flight measurements. Switching to the PLL after pull-in allows the correlators to directly match the signal's phase.

1.2.5 Extracting the Navigation Message

Every 20 ms (50 Hz) the satellite binary phase shift keys another satellite navigation message bit onto the GPS signal. If the navigation message bit changes,

there is a subsequent sign change in the output of the correlator's accumulators. The receiver's processor looks for these sign flips, synchronizes to a likely bit edge, and decodes them into a 1,500 bit satellite navigation message. The navigation message is broken into words of 30 bits (0.6 s long), subframes of 10 words (6 s long), and frames of 5 subframes (30 s long). The full satellite almanac data is distributed over multiple frames, called a superframe, which repeats every 12.5 minutes. The processor must check the words for errors using a Hamming block code (32 total bits with 26 information bits), assemble them into subframes, and extract the message data from a packed bit field.

Once subframes one through three have been assembled, the complete ephemeris is available to the receiver. With only a rough idea of the system time (down to a few milliseconds), the ephemeris enables the receiver to calculate the satellites position and, if the receiver's approximate position is known, the azimuth and elevation from the receiver to the satellite. The azimuth and elevation can then be used to correct the satellite's signal propagation time using generic ionospheric and tropospheric models. This model can be further refined with increasingly precise coefficients that are uploaded by the control segment and inserted into subframe four.

1.2.6 Taking Measurements

A bank of channels is usually referred to as a GPS correlator (Figure 1.8). The correlator takes an instantaneous sample of each channel's code phase, code DCO phase, and the carrier phase. This instantaneous sample, sometimes called a 'snapshot', contains the timing information to make a precise measurement of how far away each satellite under observation was at the moment of measurement.

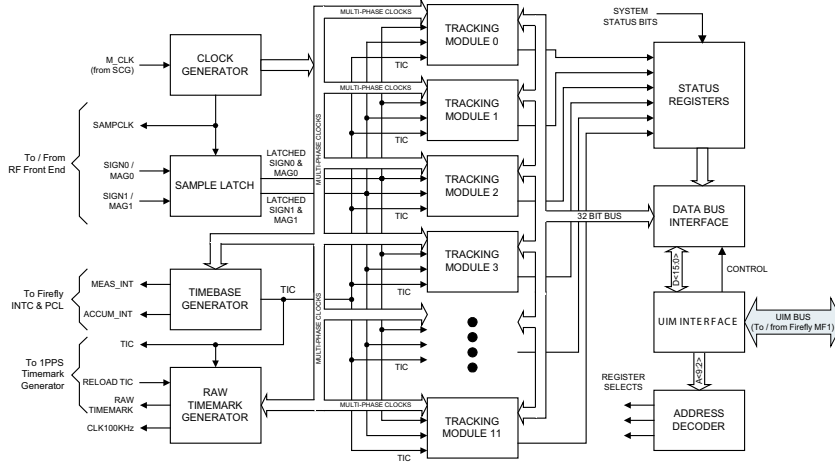


Figure 1.8: Zarlink GP4020 correlator peripheral block diagram.

Since all the satellites transmit the navigation messages at precisely the same time, the variation in the received message's total phase is a measure of the signal propagation time. This phase is directly measured as the bit number of the message, the chip number of the Gold code generator, and the phase of the Gold code generator. For example, with the Zarlink GP4020 the satellite vehicle time is calculated as:

$$t_{tx} = (B \times 0.2 \text{ s}) + (G \times 1 \text{ ms}) + (C \times 488.758 \text{ ns}) + (\phi_c \times 477.3 \text{ ps}), \quad (1.1)$$

where t_{tx} is the time of transmission from the satellite, B is the number of 20 ms long bits since the the start of the GPS week (weeks are the largest time epoch in the GPS and begin every Sunday morning at 00:00), G is the number of 1 ms Gold code cycles since the last data bit transition, C is the number of Gold code chips since the last cycle (counted in half chips), and ϕ_c is the phase of the code frequency DCO which has 1,024 counts per half chip.

1.2.7 Calculating Pseudoranges

The local clock of the GPS receiver is set to within roughly 100 ms of the system time after receiving any valid navigation message subframe, since all subframes are timestamped from the beginning of the GPS week. Given this rough synchronization, the receiver can now calculate the “pseudorange” to the satellite, which is the apparent time-of-flight of the signal measured without a precisely synchronized clock:

$$\rho_i = t_R - (t_T + \delta t_i) * c - b, \quad (1.2)$$

where ρ_i is the pseudorange from the satellite to the receiver measured in meters, t_R is the time of the measurement, t_T is the time of transmission based on equation (1.1), δt_i is the satellite clock correction factor (including terms for satellite clock drift correction, atmospheric modeling, and relativistic corrections for the satellite’s motion), c is the speed of light in a vacuum, and b is the unknown clock bias expressed in units of distance.

With four pseudoranges known, the receiver can solve for the three position coordinates as well as the clock bias term. The clock bias term is applied to the receiver’s local clock, precisely synchronizing it with GPS time.

1.2.8 Calculating Position and Time

In order to use the four pseudoranges for positioning, the location of each satellite must be known. This can be calculated to sub-meter accuracy using the precise ephemeris data obtained from the navigation messages and a standard orbital model.

When four pseudoranges and four satellite positions are available from a given correlator measurement, a position-time solution can be found using a least squares method. The fundamental positioning equation is a simple statement of Euclidean geometry:

$$\rho_i = \sqrt{\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_r} + b, \quad (1.3)$$

where ρ_i is the pseudorange from the receiver location $\vec{\mathbf{x}}_r$ to the location of the i_{th} satellite $\vec{\mathbf{x}}_i$. Like equation (1.2), the unknown clock bias b has been included.

Although straightforward in principle, this equation has several problems in application. The first is the need to invert it and solve for $\vec{\mathbf{x}}_r$. The resulting non-linear vector equation is usually linearized around an estimated solution, and then iterated until the error drops below a tolerable accuracy.

1.2.9 Carrier Phase Velocity and Position

Because the carrier frequency DCO is locked in phase with the incoming signal, a very accurate measurement can be made of the change in phase of the GPS carrier. This change in phase is the number of 19 cm wavelengths of the 1.57542 GHz L1 carrier passing per unit time, and is an independent measurement of the relative velocity between satellite and receiver. Tracking carrier phase, or “accumulated delta range”, is a very accurate method of determining receiver velocity [15].

If the integer number of carrier cycles between the satellite and the receiver can be determined, then carrier phase can be used for precise positioning. Also known as the integer ambiguity problem, solving the integer carrier phase cycles usually requires position aiding such as differential GPS (discussed below in Section 1.3.2) for single frequency receivers.

1.2.10 Filtering Position Output

Covering the gamut from simple double differenced values through Kalman filtering, position filtering allows the receiver to reduce sample-to-sample errors in position. Filtering is an entire extended GPS topic in itself, and will not be covered in this thesis.

1.2.11 GPS Receiver Software

To further understand the operation of a GPS receiver, it is helpful to break down the previous receiver description into a series of generic tasks that the processor must perform independent of the underlying hardware (see also Figure 1.9):

Tracking task — a high repetition rate, (< 1 ms) high priority task that reads the accumulators and adjusts the DCOs to acquire and track the satellite signals for each channel. This task also demodulates bits from satellite navigation messages.

Satellite navigation message task — a medium frequency, medium priority task which assembles the bits from the satellite navigation message found by the tracking task. The message is checked for errors, assembled into subframes, and the ephemeris and almanac data extracted.

Positioning task — a low frequency, preemptible task that uses measurements and ephemerides to calculate position and time.

Housekeeping task — a very low frequency, low priority task which takes care of miscellaneous chores, such as allocating satellites to correlator channels, managing external communication channels, etc.

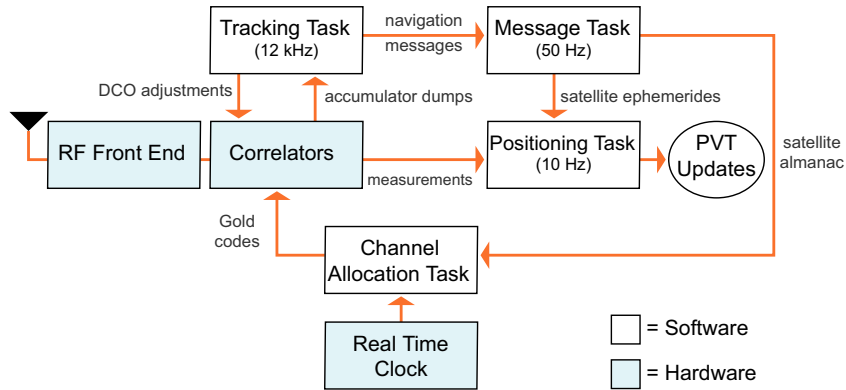


Figure 1.9: Generic GPS receiver tasks.

1.2.12 GPS Receiver Performance

Current GPS receivers have horizontal position accuracy of around 10 m (1σ) [15]. The spherical error probable (the 3D sphere in which 50% of the position estimates falls within), or SEP, is also 10 m. The system “user range error” (URE) error budget is shown in Table 1.1.

Ionospheric refraction and multipath errors are the main error sources in a modern GPS receiver. Antenna design and correlator timing can help with multipath signals, and models of the ionosphere can help subtract some of the ionospheric delay errors. To completely take into account these effects, however, requires external aiding (see Section 1.3.2).

1.3 GPS Receivers: Applications

Commercially available GPS receivers fall into general categories, based on the application’s demands.

Error Sources	1σ User Range Error (URE) [m]
Space Segment	
Clock Stability	3.0
L-Band phase uncertainty	1.5
SV parameter predictability	1.0
Other	0.5
Control Segment	
Ephemeris prediction and model	4.2
Other	0.9
User Segment	
Ionospheric delay	2.3
Tropospheric delay	2.0
Receiver noise/resolution	1.5
Multipath	1.2
Other	0.5
TOTAL (RSS) URE	6.6

Table 1.1: GPS error budget (from [15]).

1.3.1 Standard GPS Applications

Commercial GPS receivers are usually intended to provide position, velocity and time (PVT) updates once a second in an ASCII encoded message. Other data, such as satellite ephemeris and pseudoranges, are difficult, or impossible, to obtain from standard commercial receivers, and if available, are usually formatted in a propriety binary protocol. The ASCII encoded message, usually a National Marine Electronics Association (NMEA) standard, is either stored for future processing or is used by an external processor. These standard 1 Hz PVT messages are applicable for low dynamic conditions such as handheld receivers, in-vehicle navigation aides, and most commercial aircraft receivers.

1.3.2 Specialized GPS Applications

Non-standard applications require specialized code and, often, specialized receiver hardware. There are many off-the-shelf commercial receivers that are designed for specialized applications:

Timing Specialized GPS receivers exist which are intended only to synchronize a local precision clock with GPS time. Such time transfer systems can be within a few tens of nanoseconds of GPS time, which is kept within a microsecond of Coordinated Universal Time (UTC), plus or minus an integer number of leap seconds.

High Dynamics Most GPS receivers have an internal model of their dynamics; e.g., they have filtered position and velocity estimates which assume bounded dynamic movement (acceleration and velocity) and smoothness assumptions. These assumptions may cause the receiver to lose lock in high dynamic environments. High dynamic receivers, meant for military aircraft or launch vehicles, cost upwards of thousands of dollars and often need to be finely tuned for the end vehicle's dynamics.

DGPS A stationary receiver at a surveyed location can derive propagation-time corrections for each visible satellite to compensate for ionospheric and tropospheric delays. These correction signals, called “differential GPS corrections” (DGPS) can then be broadcast to local (within 100 km) roving receivers. These pseudorange corrections can give standard GPS receivers sub-meter accuracy.

RTK DGPS coupled with carrier phase positioning (carrier phase tracking with

the integer ambiguity solved) is often called “real time kinematic” positioning and is used for centimeter-level accurate land surveying.

E911 Cellular phones with embedded GPS receivers enable position-based emergency services (as well as covert surveillance) with extremely low satellite signal strengths (e.g., -160 dBm), which enables these services to be used inside buildings. The GPS time and ephemeris are transmitted to the chipset from the cellular base station, allowing the correlators to be directly set without having to search for signals. The provided ephemeris data also means the correlators do not have to decode the satellite navigation messages, letting them instead concentrate on tracking the weak signals.

1.3.3 Extended GPS Applications

Many specialized GPS applications require custom modifications to the receiver software. Although some off the shelf products may work in these applications, they are usually extremely expensive, forcing most non-governmental users to create their own solutions. Some of these applications are:

Attitude Determination Multiple GPS receivers with multiple antennas on a vehicle can be coordinated to produce an estimate of the heading (attitude) of the vehicle. This requires coordinated carrier phase tracking, where multiple receivers track the change in carrier phase caused by satellite and receiver motion.

Integrated Navigation Systems Integrating inertial, magnetic and other position and attitude sensors via data fusion (e.g., Kalman filtering) can greatly enhance the receiver’s ability to track and improve positioning accuracy. This

integration is almost always a custom solution that cannot be purchased off the shelf, with the exception of military systems which cost hundreds of thousands of dollars. Small unmanned aerial vehicles (UAVs) can greatly benefit from these kind of integrated navigation systems.

Terrestrial Physics A tremendous amount of information about the troposphere and ionosphere can be gleaned by observing how the GPS signals propagation through the atmosphere. This can be done by running the system “in reverse” — a stationary receiver in a known position with precision satellite ephemerides calculates the signal delay in the atmosphere, which is a function of the total electron content in the atmosphere along with second order effects such as water content.

Academic Applications Academic research often requires stand-alone receivers running custom software. A programmable receiver board makes an excellent teaching tool, and is necessary for in-situ prototyping and testing of algorithms.

While there is little need for a GPS development system for standard GPS applications, there is no current solution that reduces the barriers to entry for extended applications. It is these extended GPS applications — and those yet to be developed — which will most benefit from an inexpensive and open GPS development system.

Chapter 2

Motivation and Principles behind GPL-GPS

This chapter describes the motivations and general design principles behind GPL-GPS.

2.1 Motivations for an Open GPS Development System

2.1.1 The Embedded System Development Model

A software developer starting a development project on a new embedded processor generally does the following:

1. Orders a hardware development board, costing a few hundred dollars.
2. Downloads the specifications of the chipset and the development board, which is usually published on the Internet.
3. Chooses a software development system, most likely one they already own

and use. Often example code is freely downloaded from the chip manufacturer's site, or increasingly often, from independent sites on the Internet devoted to sharing technical information.

4. Develops software for which there are no intellectual property restrictions. They may share their code as they wish.

Unfortunately, developing for GPS chipsets is not as seamless. A software developer starting a development project on a commercial GPS chipset — in this example, one of the more popular chipsets — does the following:

1. Orders a hardware/software development kit for more than US \$20,000.
2. Requests specifications on the chipset and development board, forcing them to sign a non-disclosure agreement prohibiting them from sharing their work on the chipset with others — even coworkers — who have not signed the NDA.
3. Requests development tools, which are usually expensive, proprietary tools with libraries or code bases that only have an application programming interface (API) to a binary program, rather than source code. Obtaining the source code is another expense and license agreement.
4. Develops code for the new GPS chipset, which they may not share with others given their NDA. Thus, the developer is forced into “reinventing the wheel” for their application since no software is available except some generic binary interface provided by the chipset manufacturer.

Why this is standard for an industry trying to sell chipsets — not development systems — baffles the GPS developer community. Developers assume that the

chipset manufacturer is there to help, not hinder their use of a chipset. These restrictions also mean that the academic community has been all but barred from developing a teaching infrastructure because of the intellectual property restrictions on the development software. Clearly, a more streamlined and open GPS development environment, in line with the embedded development model, is needed.

2.1.2 Debugging Critical Flight Hardware

Background: PSAS

In October 2000, I directed a team of students that designed, constructed and launched an advanced sounding rocket avionics system for the Portland State Aerospace Society [16]. The vehicle, dubbed launch vehicle number 1b, or LV1b, flew to 3.6 km with a custom RISC microcontroller-based flight computer, a custom micro-electro-mechanical (MEMs) strap down inertial measurement unit, telemetry and uplink radio links, and a commercial GPS receiver board. The data from the sensors (including the GPS) was sent down a 900 MHz 19.2 kbps telemetry system and also stored in battery-backed up SRAM.

PSAS LV1b GPS Flight Data

The commercial GPS receiver's standard 1 Hz position data from the October 2000 flight is shown in Figure 2.1. The receiver was locked and positioning correctly until the vehicle reached apogee and the recovery parachute was deployed. It was extremely disheartening to realize that after the recovery system deployment shock, the receiver gave grossly incorrect position data while asserting "position locked and valid" flags. Multiple calls to the manufacturer went unheeded, and there

was no access to the receiver’s software to discover what had happened. Later discussions with other GPS developers led to the hypothesis that the errors were due to a software bug in the receiver’s Kalman filter. The need to fix this bug, along with the desire to create an integrated navigation system (GPS and inertial sensors), led us to realize that we needed to develop our own GPS software.

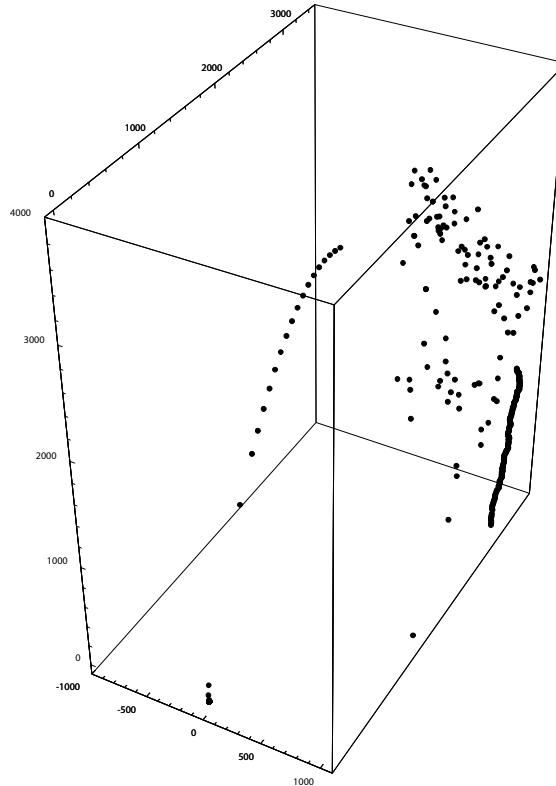


Figure 2.1: PSAS Launch Vehicle No. 1b (LV1b) GPS flight data from October 2000

2.2 Moving Towards Open Source Solutions

This began my investigation into GPS software development. It became abundantly clear that no commercially available receiver was suitable for our project,

and that no existing development system fit our needs due to budget and license restrictions.

We discovered, and became inspired by, Dr. Clifford Kelley's work on the OpenSource GPS (OSGPS) project. Kelley took an inexpensive commercial GPS receiver, "hacked" off the processor, and connected the correlator chip directly to a 486 PC using an ISA prototyping board. Kelley's work, published in 2002 [9], was the first open source GPS receiver software written. Although his receiver system couldn't easily be used in a power, space, and weight constrained environment like a launch vehicle, it did provide a code base from which to start an open source "non-hacked" receiver project.

In 2002, I proposed creating an open source GPS receiver project designed for OEM GPS boards. To my knowledge no one had yet proposed this. I began part time work on the idea starting in early 2003. I widely published the idea on the web and through the open source GPS mailing list which I formed in mid 2004. In July of 2004, Takuji Ebinuma started a parallel project by taking Kelley's OSGPS code and porting it to a commercial receiver. Although there were several problems with the project, his work became the world's first open source stand-alone GPS receiver board. Work began in earnest on GPL-GPS in late 2004, and the GPL-GPS's first position fix was on May 2nd, 2005.

Over the years, GPL-GPS was transformed from just open source software for GPS receivers into a complete GPS receiver development system. By encompassing open source methodologies, tool suites, operating systems, and open hardware designs, the GPL-GPS development system promises to provide an affordable and open GPS development environment available to all.

2.3 GPL-GPS 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 by multiple end users with different needs. Linux is a prime example of such an open system; thousands of people around the world have embraced and extended Linux from a small educational project to a modern, full featured desktop and enterprise-level operating system. GPL-GPS has five design principles:

Open Source: All software used in this project must be open source. This includes the application code, any operating system, and the 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.

Portable: The software design should be as modular and portable as possible; improvements to GPL-GPS code should be transferable to future GPL-GPS ports, as well as back to the OSGPS project. The development system should run on as many platforms (Linux, Windows, Macintosh) as possible.

Inexpensive The hardware cost of the system must be as low as possible (a few hundred dollars, with a US \$300 upward limit) and be widely available.

Available All software and documentation must be made available on the internet. Any developer with a bit of experience who is willing to read the

documentation should be able to quickly and easily start development.

The GNU General Public License (GPL) was chosen as the software license because of its general acceptance as an open source license. Some industry lobbyists claim that the GPL itself is a restrictive license agreement since any source changes must be distributed along with the binary form of the software, making it impossible to have proprietary source code based on the original open source code. The free software community claims that this restriction isn't a restriction, but rather a requirement for participation. People who benefit from an open system should expect that their modifications should be open as well, if they choose to distribute them.

Note that licensing under the GPL does not require that independently developed code be distributed openly. Also, code that simply interacts with GPL-licensed code does not fall under the GPL [8].

2.4 Immediate Applications for GPL-GPS

GPL-GPS is primarily aimed at small academic and/or commercial projects which require extended GPS applications. For example:

UAVs Small unmanned aerial vehicles (UAVs) requiring integrated navigation systems such as the PSAS' next generation launch vehicle, LV2.

Robotics Autonomous robotic navigation includes both autonomous agricultural equipment and autonomous vehicles such as those entered in the Defense Advanced Research Project's Grand Challenge competition.

Nanosatellites GPS receivers on satellites make satellite navigation and attitude control smaller and less expensive compared to larger, more costly star sensors. Several academic nanosatellite projects have already expressed interest in using GPL-GPS.

GPS coursework GPL-GPS should make a dramatically inexpensive and available educational development system.

Chapter 3

Selecting a Receiver Chipset and Board

3.1 Choosing a GPS Receiver Chipset

A GPS “chipset” is the active electronics necessary to decode GPS signals. Most GPS receivers require one to three silicon integrated circuits (ICs, or ‘chips’): a radio frequency (RF) front end, a baseband correlator, and some kind of general purpose processor. In practice, commercial GPS chipsets come in four configurations:

Correlator only A single chip that only does GPS correlation. It requires a separate RF front end and processor.

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

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

Analog to Digital Converter (ADC) Software-defined radio (SDR) receivers replace the RF front end and correlator chips with a single ADC and signal processing software running on a fast processor.

Current trends seem to be bifurcating the GPS chipset market: the system-on-chip (SoC) trend is pushing receiver chipsets towards highly-integrated single chip receivers while, at the same time, trends in software defined radio (SDR) are pushing the correlator features into small “correlator peripheral” chipsets that use the processing power of a host processor for much of the signal processing besides tracking. The latter is most evident in deeply embedded receivers, such as those in GPS-enabled cellular phones.

3.1.1 GPL-GPS Chipset Requirements

The GPS chipset used in the GPL-GPS receiver will determine how the initial code infrastructure is defined, and deeply influences how project resources are spent. Does the chip need to be reverse engineered? Is that even possible? Are there operating systems already ported to the processor of the chipset? Choosing a chipset sets the tone and direction for the rest of this project, thus considerable research was invested in existing chipsets.

It can be argued that an open source GPS receiver should move away from a hardware-dependent GPS chipset towards a SDR system. This would increase the flexibility of the receiver’s functionality while reducing the the cost and complexity of the receiver hardware. Unfortunately, moving from a GPS correlator chipset to a general purpose processor or field programmable gate array (FPGA) currently incurs unacceptable power and size penalties for many embedded applications:

chipset-based receivers generally require 6 in² and consume less than 1 W, while general purpose processors and FPGAs require at least 16 in² and consume 10–100 W of power. Thus to serve power, weight and size restricted applications, GPL-GPS must use a chipset-based receiver rather than a SDR general purpose platform. As a stand-alone receiver, the chipset for GPL-GPS requires:

1. Open and accessible chipset documentation (it cannot require a non-disclosure agreement)
2. Support of the processor architecture (if there is one) by open and available tools.
3. Available in a commercial, off-the-shelf (COTS) receiver board.

Important but not required features of that chipset are that it:

1. Has at least eight correlator channels.
2. Is intended to be used as a general purpose receiver and is not as an ultra-miniaturized SoC cell-phone chip or a timing-only chip.
3. Has a convenient and usable hardware architecture (i.e., does not require an extraordinary amount of effort in either software or hardware development).
4. Has an existing open code base

3.1.2 Applicable GPS chipsets

There are currently about a dozen commercial GPS chipsets on the market. Of these, most were not user-modifiable or programmable as a stand-alone receiver and thus were discarded. Discarded chipsets included:

- Older chipsets with significantly less processing power or less than 8 correlator channels.
- “Position peripheral” chipsets, meant for deeply embedded consumer applications such as cellular phones. In such applications, a receiver chip provides a standard interface over a parallel or serial link to the host processor. The positioning peripheral chip — despite the fact that it may contain a general purpose processor and may even require the host to send it a firmware — must be considered to be non-programmable since only its application interface is described. Data sheets on the internal workings of the chipsets do not seem to be available.
- Intellectual property-only (“IP-based”) receiver designs, consisting mostly of hardware description language files for Field Programmable Gate Arrays (FPGAs), were not considered for GPL-GPS. As mentioned earlier, FPGA boards are larger and consume more power than conventional GPS chipsets, and further, as of fall 2004, there were no commercial FPGA-based GPS receiver boards available.

Table 3.1 compares available chipsets that fit the stand-alone receiver model. The chipsets have been divided into two groups: the first has integrated processor and correlator chipsets, and the second group has separate process and correlator chipsets.

Unlike the rest of the semiconductor industry, finding an “open” GPS chipset — one which had open and available documentation and did not require a restrictive licensing agreement to use — was a major challenge. Exhaustive research uncovered only two open chipsets. This seems like extreme short-sightedness on

Mfg. and model.	Arch.	Open	Rec	Sane	OSS
Atmel ATR0620	ARM7TDMI	N	Y	?	N
NemeriX NJ1030 [14]	SPARC V8	Y	N	Y	N
SiRF SiRFStar II [17]	ARM v?	N	Y	Y	N
Thales Baldur [19]	ARM7	N	Y	?	N
u-Nav uN8031B [22]	V-DSP?	N	Y	N	N
Zarlink GP4020 [25]	ARM7TDMI	Y	Y	Y	Y

Processor with integrated correlator peripheral chipsets.

Mfg. and model.	Open	Rec	Sane	OSS
Navman Zodiac [13]	N	Y	Y	N
Trimble FirstGPS [20]	N	Y	?	N
Zarlink GP2021 [23]	Y	N	Y	Y

Correlator only chipsets.

Key:	Arch	Architecture of processor if known
	Open	Open documentation available without NDA or other excessive licensing agreement
	Rec	Inexpensive, commercial GPS receiver boards exist for this chipset
	Sane	Receiver architecture is not overcomplicated
	OSS	Open source software exists for this chipset

Table 3.1: Comparison of GPS chipsets.

behalf of the GPS chipset manufacturers who are operating on an older model of intellectual property retention than the more open, fluid — and successful — general semiconductor industry.

I spent some time trying to convince chipset manufacturers to open up their chipset designs, if only for nonprofit projects. All refused, and when queried, answered that they were protecting their intellectual property from competition. In light of the openness and success of the general semiconductor industry, this seemed to be poor business strategy. The only explanation offered by other frustrated GPS developers is that the military origins of the Global Positioning System have left a corporate culture still steeped in secrecy and closed intellectual property rights.

3.1.3 Choosing a Chipset

The Nemerix NJ1030 [14] is an open GPS chipset featuring a 16 channel correlator, an open source “LEON” (SPARC V8) processor, a large onboard SRAM cache, and very low power consumption (< 8 mA at 3.3 V). Unfortunately, this promising new chipset was too recently released to have any commercially-available receivers as of spring of 2005. Future versions of GPL-GPS will undoubtedly be ported to the Nemerix chipset as NJ1030-based receivers begin to appear on the market.

The only open and available chipset at the start of the GPL-GPS project was the Zarlink GP4020 Baseband Processor [25]. All architectural and application notes are available on Zarlink’s web site, including reference designs for receiver boards. A major feature of choosing the GP4020 is that its correlator peripheral is the same architecture as the stand-alone Zarlink GP2021 correlator chip used in Kelley’s OpenSource GPS receiver project. This provided a rich pre-existing soft-

ware base for any GP4020 design. Another plus of the GP4020 is that its general purpose processor is an ARM7TDMI, a widely used 32 bit RISC architecture with available open source tools.

The GP4020 has a variety of peripherals, including two timers, two asynchronous serial ports (UARTs), a high speed synchronous serial port (SPI), a watchdog timer, and a sophisticated memory peripheral controller (MPC) to interface the GP4020 to external RAM, ROM and peripheral devices. A typical GP4020-based receiver is shown in Figure 3.1. Note the antenna, TCXO, and filters on the GP2015 RF front end, and the 2 bit ADC output (sign, magnitude) from the RF front end to the GP4020 correlator block. The 16 bit RAM and ROM (usually flash) is shown in the upper right.

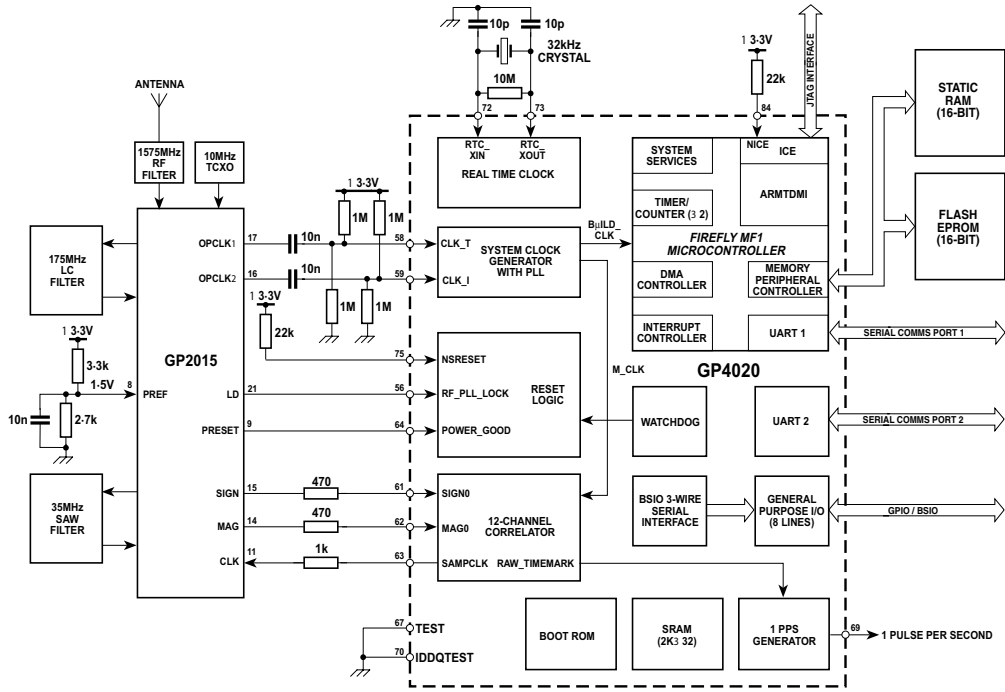


Figure 3.1: Generic GP4020-based GPS receiver block diagram (reproduced from [25]).

3.2 Choosing a GP4020-based GPS Receiver

3.2.1 GP4020-based GPS Receiver Requirements

The GPL-GPS receiver requires a receiver board that:

1. Uses the Zarlink GP4020 Baseband Processor chip.
2. Has enough RAM and ROM (> 128 kB).
3. Does not need to be modified in any way to be used as a GPL-GPS receiver.

This last requirement is important because many developers do not have the skills necessary to precisely modify a four or more layer printed circuit board. Relying on “hacking” a receiver board severely diminishes the number of software-only developers willing to work on a project. Other important considerations for the receiver are:

1. Access to GP4020 pins, in particular the MULTIFNIO pin for easy use of the GP4020’s built-in boot loader and the GPIO pins for LED debugging.
2. Compact size and on-board power management.
3. Standard RF connector (e.g., MCX, SMA, SMB)
4. Widely available.
5. Relatively inexpensive (less than US \$150).

3.2.2 Available GP4020-based GPS Receivers

See Table 3.2 for a comparison of the three commercially available boards based on the GP4020.

Manufacturer	SigTec	SigTec	Novatel
Model	MG5001	MG5003	SuperStar II
RF Front End	GP2015	GP2015	GP2015
External SRAM	256 kB	512 kB	128 kB
Flash	256 kB	256 kB (serial)	256 kB
Cost (Qty. 1)	US \$250	US \$225	US \$125
Notes	51 pin connector	Small form factor	32 kB EEPROM

Table 3.2: Commercially available GP4020-based GPS receivers.

3.2.3 Choosing a Receiver

The SigTec MG5001 is the clear choice as a development receiver for a number of reasons:

- Larger RAM size
- 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 (UARTs), JTAG debugging pins, MULTIFNIO serial boot pin, GPIO pins, data lines and a small subset of address lines.
- The MG5001 has a RF section that is separated from the digital section in a way that facilitates using separate metal ground shields, a useful option when operating in high noise applications.

The MG5003 has the advantage that it has an extremely small form factor and has a large amount of SRAM. But the lack of connectors make it necessary to hack the board to get access to the GPIO, and the serial flash make it impossible to run the software infrastructure directly from flash, making the larger SRAM not as advantageous.

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 SigTec 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. With the exception of RAM size, code developed on the MG5001 should work exactly the same on the SuperStar II. Projects with budget constraints should still be able to use the SuperStar II for GPL-GPS development. In-situ GPL-GPS installations should also consider using the SuperStar II due to the lower cost, since only development activities require a large SRAM size.

An important possible difference between the boards is the sensitivity of their RF front ends. While they employ the same RF front end chip (the Zarlink GP2015), they have very different component values and board layouts. Future side-by-side testing using a single antenna with a 2-to-1 splitter will enable a direct comparison of the two receiver boards.

3.2.4 Creating a GPL-GPS Development Board

It's almost always necessary to have some kind of a development board that aids embedded development work. To have something similar for GPL-GPS, I designed a 4.5 x 3.5 inch carrier board for the MG5001. It includes:

- Two DB-9 connectors with a 5 V to RS-232C signal level converter chip, allowing a PC to connect to both serial channels.
- Three switches connected to the GP4020's reset, MULTIFNIO (serial boot-loader), and JTAG debugging pins.

- Five separate 0.1 inch headers for the address, data, JTAG, GPIO and serial pins.
- Eight LEDs connected to the GPIO.
- Linear 5 V and 3.3V power supplies.
- A small prototyping area.

The schematic and board layout was created in EAGLE CAD, a freely available (but not open source) PCB CAD development tool. The design is posted on the GPL-GPS web site (<http://gps.psas.pdx.edu>) for others to download and manufacture. 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. See Appendix B for a detailed schematic.

Chapter 4

Selecting and Porting a RTOS

While it was possible to write a custom operating system, or even use a simple C language run-time environment for GPL-GPS, neither allows the developer the flexibility to focus on the application, rather than the underlying code infrastructure. The open systems approach of GPL-GPS also suggests using an already existing standards-based, open software system like an operating system. Thus, I chose to use a standard real time operating system as the base of GPL-GPS' software infrastructure, rather than the seemingly more typical GPS development option of creating a custom solution.

4.1 'Soft' vs. 'Hard' Real Time

Operating systems are grouped into three general time-constrained categories: non-real time, 'soft' real time, and 'hard' real time operating systems. A standard, or non-real time, operating system has no bounds on system response and can not run applications with time constraints. Soft real time systems have low latencies, but still have no bounds on system response time. In many respects, 'soft real time' is a misnomer — if a system can miss deadlines without a failure, then it's

not a real time system. Hard real time systems will fail if their time constraints are not always met. For these systems, being late is similar to a logical error.

Real time systems have exacting time constraints that must be met. These constraints may be measured in microseconds, hours, or even days, but they must be met.

GPS receivers are an example of a real time system with critical timing constraints on the order of 100's of microseconds: losing the 1 ms correlator accumulator interrupts will eventually cause the channel to lose lock on a satellite.

4.2 RTOS Requirements

The GPL-GPS project has very strict requirements for a RTOS because of the performance and memory constraints. The requirements for the RTOS are:

Hard real time performance Interrupt latencies must be on the order of $10\ \mu\text{s}$ in order to handle the 1 ms accumulator interrupts, and context switches must be on the same order of magnitude to enable the use of threads.

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 philosophy, the RTOS must be open source with no licensing restrictions.

Existing port to ARM7 processors Porting a RTOS to a new processor architecture is a thesis unto itself, so the RTOS must be already ported to ARM7TDMI processors.

Integrated debugging tools Built-in debugging features are one of the most overlooked aspects of an operating system. In order to make cross-platform embedded development tolerable, the RTOS must support remote loading and debugging of applications on the target system.

Some important considerations for choosing an RTOS include:

Rich operating system primitives A benefit of using an operating system is not only the ubiquitous support of threads, but also that it provides rich timer and interprocess communication (IPC) primitives (e.g., mutexes, semaphores, etc).

Simple to compile and configure The more complicated the setup and installation of an OS, the harder it is to have others adopt it and the more time wasted on the code infrastructure rather than the application code.

Supports multiple architectures An RTOS which is ported to many other processor architectures is an important consideration for future GPL-GPS ports.

Good documentation Poor technical documentation on the complicated inner details of an operating system can be frustrating and waste time. Books, online manuals, and manuals all help clarify the details of using a RTOS.

4.3 Applicable Real Time Operating Systems

Several months were spent research existing real time operating systems. The comparison of the applicable operating systems are in Table 4.1.

μ Clinux was discarded since it is not a real time operating system. Nucleus turned to out to be thousands of dollars per application, and not open source, which

RTOS	ISOS [11]	μ C/OS-II [12]	eCos [7]	μ Clinux [5]	Nucleus [1]	Custom
Hard real time	Y	Y	Y	N	Y	Y
< 128 kB footprint	Y	Y	Y	N	Y	Y
Open source	Y	N (published)	Y	Y	N	Y
Supports ARM7TDMI	Y	Y	Y	Y	Y	Y
Built-in debugging	N	N	Y	Y	Y	N
Rich IPC features	N	Y	Y	Y	Y	N
Simple configuration	Y	Y	Y	Y	Y	N
Supports most proc.	N	Y	Y	Y	Y	N
Existing documentation	N	Y	Y	Y	Y	N

Table 4.1: Comparison of real time operating systems.

is unfortunate since it's a truly full featured, configurable and light weight RTOS. ISOS was small and simple, but wasn't feature rich and had no debug facilities. As mentioned earlier, a custom-written RTOS was considered, but rejected because of the amount of work necessary. Further, a custom RTOS solution would not provide the feature-rich, standards-based functionality that an existing RTOS provides. This left only $\mu\text{C}/\text{OS-II}$ and eCos. $\mu\text{C}/\text{OS-II}$ turns out to be a well written RTOS with FAA DO-178B certification and a nice thick companion textbook explaining operating system fundamentals using $\mu\text{C}/\text{OS-II}$ as an example. It was the clear choice, until it became evident that it provided no debugging facilities, and that the license agreement doesn't allow redistribution.

4.4 Choosing a RTOS

eCos turned out to be the only choice worth considering. Fortunately, it exactly fits the RTOS requirements and important considerations. eCos is:

- A hard real time operating system with low interrupt latencies (approximately $8\ \mu\text{s}$ for an ARM7TDMI at 20 MHz),
- Ported to several existing ARM7TDMI processors,
- Bundled with RedBoot, a tiny but powerful boot manager that includes GDB stubs, an open source remote debugging tool,
- Tiny enough to fit in under 50 kB of flash (although 80 kB is a more reasonable size),
- Free and open source (under the GNU GPL v2),

- Ported to more than 10 different processor architectures (including the LEON SPARC v8 processor used in the Nemerix NJ1030 GPS baseband processor)
- Well documented via an online reference manual and a published book.

Other than Nucleus, eCos has the richest feature set of the surveyed operating systems: a POSIX compatibility layer, several flavors of debug monitors, sophisticated handling of interrupt service routines, and, importantly, a very sophisticated debug and instrumentation environment. eCos has the added benefit that its required toolset is free and open source. It requires the GNU software development system that includes tools such as the GCC compiler and the GDB debugger [7].

eCos's main drawback is that it is an extremely complicated RTOS with the most sophisticated configuration tool of any comparable operating system. Indeed, it is the only configuration tool to have its own language.

4.5 eCos Architecture

eCos is not a standard real time operating system that runs independently on the target system and provides an environment to run applications. It is a “runtime system”, a static pre-compiled library that the application links against during compile time. The library provides all of the functions of a standard operating system: startup, RTOS kernel, scheduler, etc.

eCos is called the “embedded Configurable operating system” because it uses a wxWindows-based graphical configuration tool to control the features compiled into the library. Like many configuration systems, it must carefully orchestrate dependencies amongst the packages. The Atmel 29LV200BB flash memory drivers required for the MG5001 board, for example, require the generic flash support

package, which requires a series of file-system-like extensions. This web of dependencies is handled by a TCL-based language written specifically for eCos called the “configuration description language” (CDL). Users run the graphical `configtool` to select eCos packages (Figure 4.1), and then choose components inside those packages. On a source level, the configuration tool includes source packages in the eCos library and `#defines` component options.

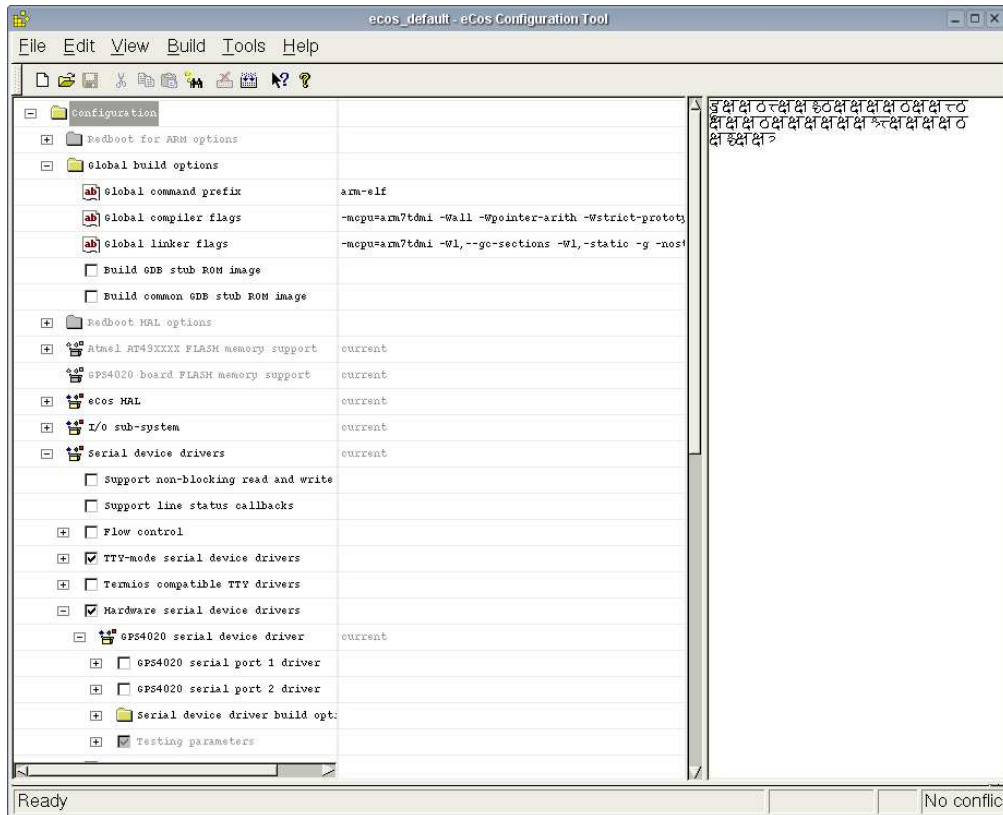


Figure 4.1: The eCos `configtool` program configuring the `gps-4020` template.

The configuration system allows eCos to be well “tuned” to the target system; full featured systems can be only 50 kB, while including larger packages, like ethernet drivers, POSIX compatibility, and the GoAhead web server can push eCos

to several megabytes.

eCos uses a hardware abstraction layer (HAL) to keep its kernel independent of the underlying system architecture. Calls that involve hardware — turning off the interrupts, for example — are `#defined` by the architecture template into direct hardware calls. This means that the kernel can be almost completely independent of the hardware. This gets more complex the closer one gets to hardware, but it maintains a reusable code base independent of the underlying hardware [10].

4.6 Porting eCos to the GP4020

eCos ports have three levels: architectures, variants, and platforms. Architecture ports are ports to new processor architectures (e.g., x86 vs ARM), variant ports are different versions of an architecture (e.g., ARM7TDMI vs XScale), and platforms are differently configured development boards (e.g., the MG5001 vs the SuperStar II receiver which have different memory sizes). Thus porting eCos to the Zarlink GP4020 baseband processor required a variant port for the ARM7TDMI-based GP4020 and a platform port for the SigTec MG5001 receiver.

The rich feature set and its custom description language makes even eCos platform ports nontrivial. After early attempts at creating a variant port from a more generic ARM7TDMI processor met with failure, Gary Thomas, one of the original authors of eCos and the original author of the ARM architecture port for eCos, completed the variant and platform port. The numerous technical details of the port will not be presented here: for more information, please see the GPL-GPS project web site at <http://gps.psas.pdx.edu/>. The GP4020 variant and platform ports includes:

- A program to allow minicom, an open source serial terminal application, to use the GP4020 serial port bootloader.
- Memory definitions and code for the GP4020's memory peripheral controller (MPC).
- Device drivers for the GP4020 serial ports, timers, and interrupt controller.
- CDL files to describe the `gps4020` eCos package.
- CDL files to describe RedBoot boot loader RAM and ROM (flash) images.
- CDL files to describe `gps4020` eCos images, including RAM and ROM (flash) images.

4.7 GPL-GPS Software Infrastructure

Figure 4.2 shows the multi-layered 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 is used in many commercial development boards, and includes such services as a simple command line interface, serial and ethernet communication with up and download protocols, flash drivers, and debugging executives like GDB stubs, a debug executive for the GDB debugger. 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.

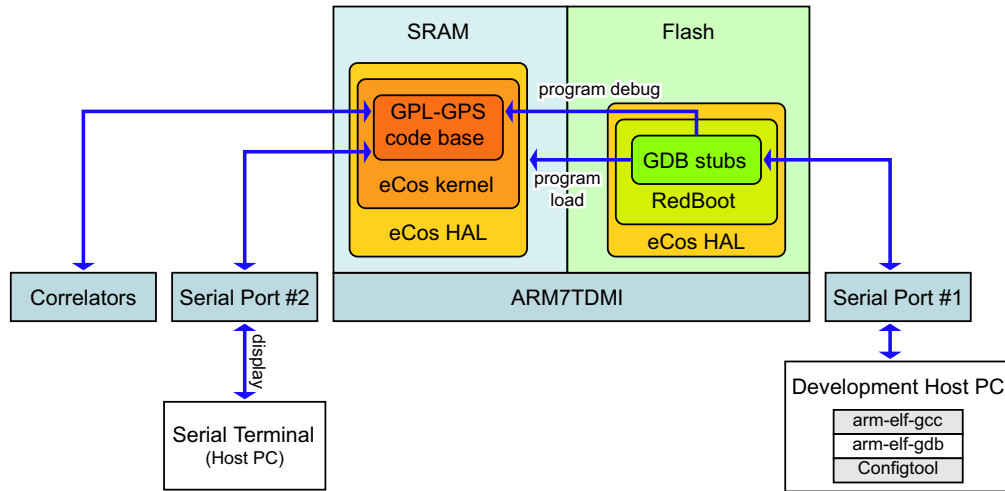


Figure 4.2: GPL-GPS software architecture: GPL-GPS, eCos, RedBoot and host system.

4.8 Getting eCos Running on the GP4020

Detailed instruction on installing RedBoot and the GPL-GPS development system can be found on the GPL-GPS website, <http://gps.psas.pdx.edu/>. In summary:

1. Compile three eCos libraries: a RedBoot executive for RAM, a RedBoot executive for ROM, and an eCos library for the GPL-GPS application.

2. Using the GP4020 bootloader and the Minicom terminal program on a host PC, load the RedBoot executive for RAM into the receiver's RAM.
3. Using the RedBoot image now running in RAM, load the RedBoot executive for ROM and write it into the receiver's flash memory.

Now the receiver has RedBoot stored in flash. At this point, the receiver is ready for application development. To use the GPL-GPS application, the user must:

1. Compile GPL-GPS sources, linking with the eCos library compiled in the previous steps.
2. Run GDB (`arm-elf-gdb` in this case) on the host PC and using GDB stubs in the RedBoot image, load the application into the receiver's RAM.
3. Run and debug the program using the remote GDB protocol over the serial port.

Chapter 5

Porting OpenSource GPS

5.1 Introduction to OpenSource GPS

With only a modified GPS receiver mounted on an ISA prototyping card (Figure 5.1), a 486-based PC, and the Borland C compiler, Dr. Clifford Kelley launched the open source GPS movement by publishing his results in 2002 [9] and then releasing his code under the GPL. Since then several projects (including GPL-GPS) have begun from his original “OpenSource GPS” (OSGPS) project.

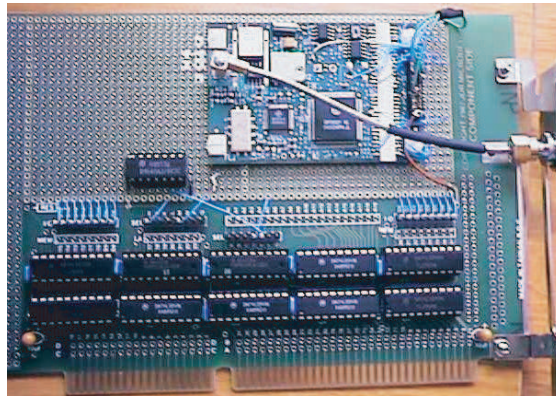


Figure 5.1: The original OSGPS ISA PC card (reproduced from [9]).

OSGPS uses a foreground/background (interrupt/mainline) architecture. The

foreground task is driven by a $500 \mu\text{s}$ timer interrupt on the host PC. All time dependent functions — accumulator dump processing, satellite navigation message decoding, and measurement processing — are called directly from a single interrupt. Navigation algorithms and display code run as the mainline task with communication between tasks accomplished using global variables (Figure 5.2).

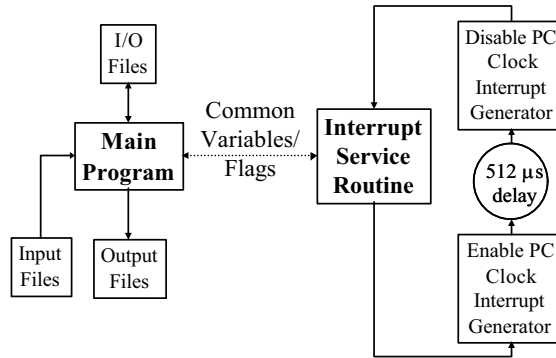


Figure 5.2: OpenSource GPS software flowchart (reproduced from [9]).

OSGPS is a ground breaking project, and as such, has several important issues:

- OSGPS is not divided into modules by GPS task, making it difficult to follow its control and data flow (e.g., compare Figure 5.2 with Figure 1.9).
- The mainline code could be better prioritized and scheduled if it were broken into threads.
- OSGPS has very few comments to explain the algorithms used, making it very hard to understand why algorithms and control flow was implemented in certain ways.
- While the algorithms used in OSGPS are time honored, they are not necessarily the most efficient or the best for any particular application. Some way

to break out the algorithms into modules for easy swapping of algorithms would be useful.

- OSGPS relies on the legacy and closed source DOS operating system, and the no longer supported Borland 4.5 C/C++ compiler.

5.2 ARMGPS: Porting OSGPS to the GP4020

In 2004, inspired by GPL-GPS, Takuji Ebinuma started his own project, ARMGPS [6]. In a few months, he ported a stripped down version of OSGPS v1.15 to the SigTec MG5001 receiver using JaysOS [3], a very simple operating system for ARM7TDMI processors. Ebinuma stripped out all carrier phase, almanac, and file handling features in the process and kept the same foreground/background task model. He attempted to break the background task into threads, but the implementation was no more effective than OSGPS' mainline task because the threads were called by timer delay rather than being event driven.

While ARMGPS does work, it is unstable, prone to dropping accumulator interrupts and to bouts of unpredictable behavior. Ebinuma attributes the instability of ARMGPS to the simple nature of JaysOS which forced a primitive thread architecture with no interprocess communication (IPC). ARMGPS also does not fully implement the changes needed to switch from the OSGPS hardware (486 PC with a GP2021 correlator) to the GP4020-based receiver. Finally, JaysOS lacks any reasonable debugging features, so tuning and debugging ARMGPS is very difficult. After struggling with the code for a few months, Ebinuma gave up on ARMGPS and officially closed the project.

5.3 Transforming ARMGPS to GPL-GPS

In late 2004, I revived the GPL-GPS project and began fine tuning the GP4020 eCos port. In early 2005 I took Ebinuma's ARMGPS code, and using Kelley's most recent OSGPS code (v1.17) as a reference, ported it to eCos. GPL-GPS attained "first fix" on May 2nd, 2005, and is now publicly available on the project's web site (<http://gps.psas.pdx.edu/>). Although never truly finished because of its development system nature, GPL-GPS now exists as a code-complete development environment for creating extended GPS applications with commercial GPS receivers.

Roughly counted by lines, OSGPS is 6,300 lines of code, ARMGPS is 5,000 lines of code, and GPL-GPS is currently at 6,000 lines. Of these 6,000 lines, an estimated more than 80 % have been restructured, refactored or replaced in the transition from ARMGPS and OSGPS to GPL-GPS. This section does not attempt to describe all the detailed code changes that have occurred. Instead, I will attempt to list the broad categories of improvements made to the legacy code base.

5.3.1 GPL-GPS Computing Environment

The changes made from OSGPS/ARMGPS to GPL-GPS must be understood within the context of the GP4020 computing environment. Typical GP4020-based receiver boards have:

- A 32 bit integer-only ARM7TDMI running at 20 MHz
- 8 kB of fast 32 bit wide internal SRAM

- 128–512 kB of 16 bit external zero-wait-state SRAM
- 128–512 kB of 16 bit external one-to-three wait state flash
- 0–512 kB of serial EEPROM

Typical throughput for these boards is roughly 5 MIPS from flash memory, 10 MIPS from external SRAM, and 20 MIPS from internal SRAM. The ARM7-TDMI has a single cycle 32 x 32-bit integer multiply, very sophisticated relative addressing schemes, and a very useful branch-per-instruction capability. Note the ARM7TDMI has a 32-bit wide RISC instruction set and a 16-bit wide alternate instruction set called “thumb” mode. Although thumb mode seems optimal for the 16-bit external memory (both SRAM and flash), it is not well supported by eCos. If future versions of eCos support Thumb mode, then it is likely GPL-GPS will begin using it.

5.3.2 Optimizing for GPL-GPS

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

GPS task-based threads Possibly the most important optimization has been repartitioning the OSGPS/ARMGPS code to reflect the various GPS receiver tasks (see Figure 1.9). Each task has been modularized into its own code and header file, to clarify and prioritize the tasks. Each task usually has one thread and one data structure associated with it, making it a more modular interface for task additions and/or inserting external communication links (e.g., an external flight computer on a UAV). See Figure 5.3 for the repartitioned task and data flow.

Event driven processing Instead of timer delays, GPL-GPS uses event-driven threads and IPC, including flags, semaphores and mutexes. This mode of operation decreases both response time and the amount of code scheduled to run at any one time.

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 and to save space in the small but fast internal 32 bit SRAM. Because of the ARM7TDMI's single-cycle 32, 16 and 8 bit memory access abilities this should not slow down processing.

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 threads which run in as a low priority, infrequent task (e.g., the position code), floating point may be left in place for coding ease and clarity.

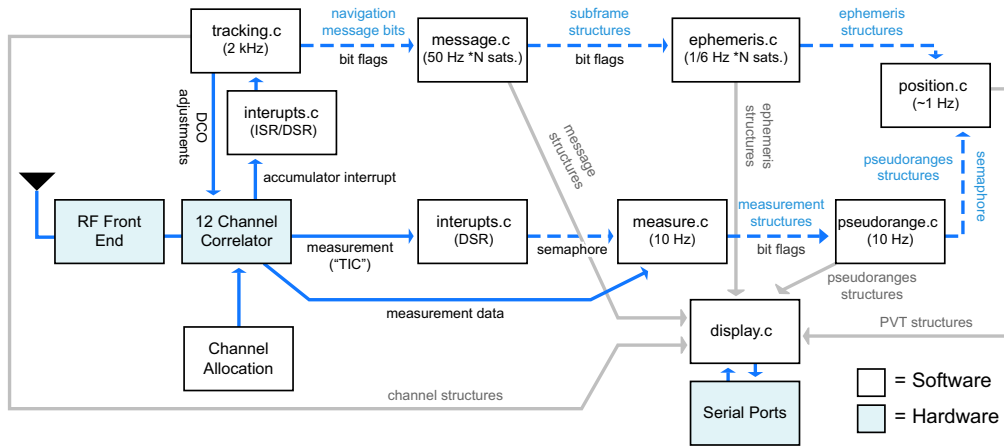


Figure 5.3: GPL-GPS software flowchart.

5.3.3 Improving on OSGPS Algorithms

GPL-GPS tries to refactor OSGPS algorithms to take a more efficient approach while improving code clarity and readability. Unfortunately these are sometimes mutually exclusive. So far the improvements include:

- Streamlined navigation data decoding, and refactored the message finding algorithm to operate on subframes rather than frames. This allows data to be grabbed every subframe period (6 s) rather than once each frame (30 s). The shorter message chunk also improves performance in noisy environments.
- Navigation message parity check now uses an improved and clearer algorithm.
- Changed hardware register access from a function-based peek-poke interface to memory-mapped structures.
- The position calculation has been re-worked to reduce the computational load and to clarify the computation that is being done.

- Timing and data flow have been improved by eliminating redundant variables and redundant code, and instituting a more consistent naming convention and coding style.
- Added an optimal cold start algorithm based on satellite orbital slot occupancy.
- Removed many mysterious constants and replaced them with well defined and documented physical or hardware defined constants. The mysterious constants are still there, but now are computed at compile time with explicit algorithms.
- Rewrote from scratch numerous helper routines improving efficiency, correctness and clarity.
- Commented and lightly refactored the tracking code.

5.4 GPL-GPS Current Status

GPL-GPS became an operational GPS development environment on May 2nd, 2005 with its first successful position fix (see Appendix A for initial positioning results). While the positioning code is currently crude — there is no atmospheric correction, carrier phase positioning, or position filtering — rapid progress can be made now that the software and development infrastructure have been completed.

5.5 GPL-GPS Current Performance

Figure 5.4 shows the timing characteristics of two time-critical GPL-GPS tasks: the tracking loops, which run every $505 \mu\text{s}$ 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 $355 \mu\text{s}$, 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 move to fixed point math. The measurement thread takes $380 \mu\text{s}$, but note that it is interrupted and suspended while the tracking loop runs.

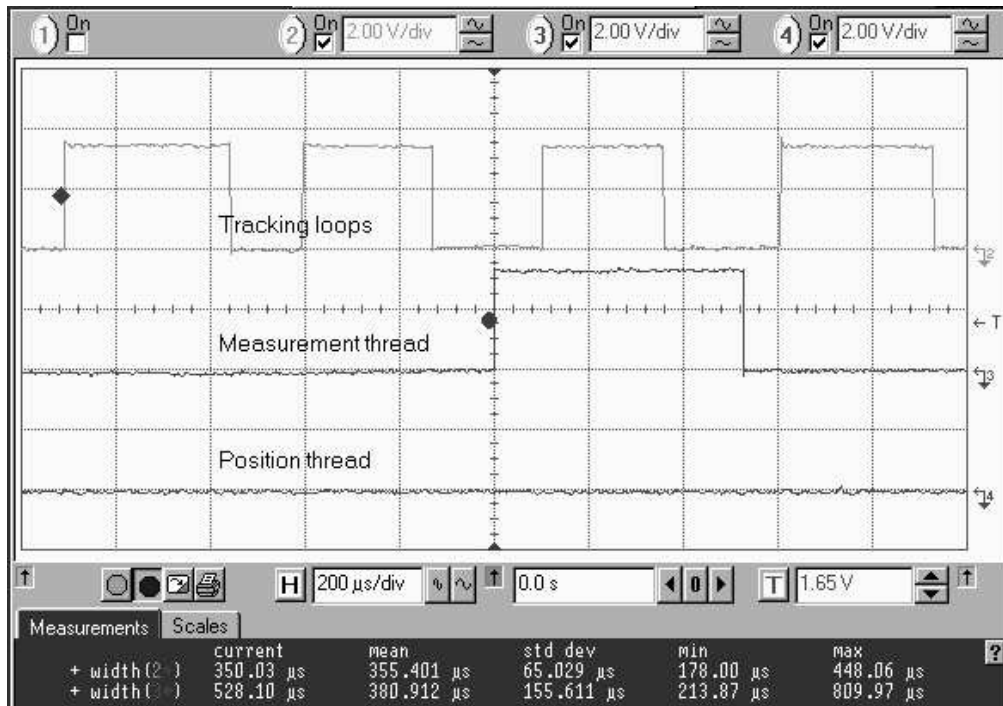


Figure 5.4: GPL-GPS timing: tracking loops and measurement thread.

Figure 5.5 shows the tracking loops, measurement thread, and the position

thread. In this figure, the position thread is only processing 3 satellites and thus is taking only 70 ms. The position thread takes between 0.3 s (for four satellites) and 0.6 s (for seven satellites) when calculating position.

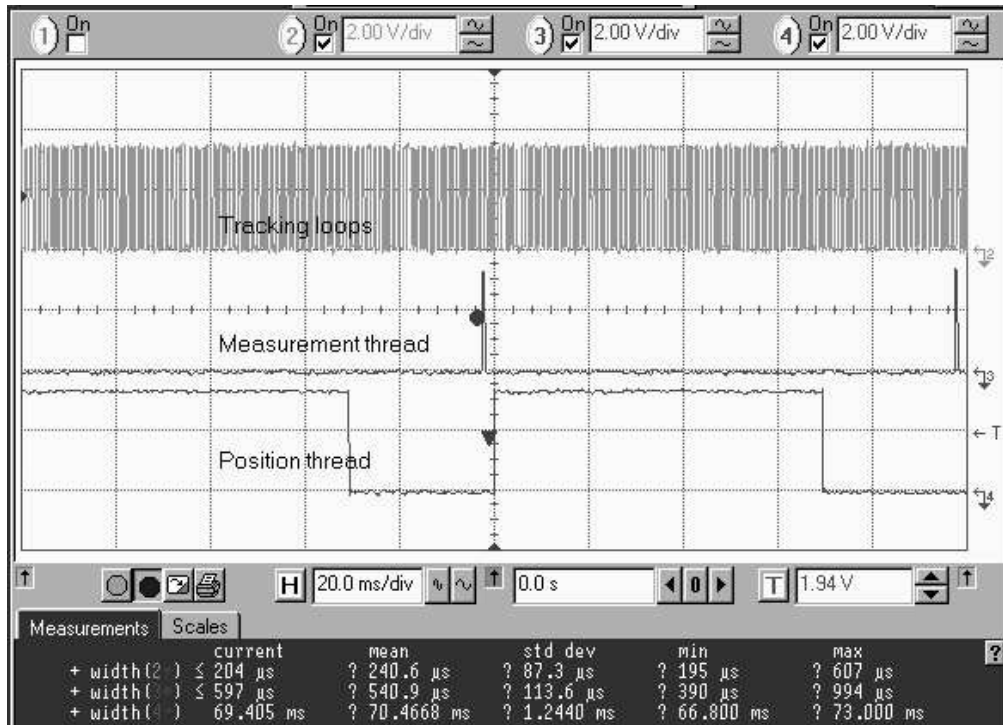


Figure 5.5: GPL-GPS timing: tracking loops, measurement thread, and position thread.

Chapter 6

Future Work and Conclusion

6.1 Future Work

6.1.1 Introduction

The potential of GPL-GPS exists in the extended applications end users can implement because of their complete access to the software and hardware of the GPL-GPS development system. Before attempting extended applications, however, several issues need to be resolved. The following “laundry list” is a prioritized list of features I, and the other contributing members of the GPL-GPS community, hope to accomplish:

6.1.2 Reimplement Atmospheric Correction

Atmospheric models of ionosphere and troposphere propagation delays were removed from OSGPS to simplify porting. This should be reimplemented, along with some refactoring of the satellite position calculation.

6.1.3 Reimplement Almanac Processing

Almanac processing was removed from OSGPS to simplify porting. This should be reimplemented, along with more sophisticated satellite-to-channel allocation algorithms based on the almanac and a driver for the GP4020 real time clock (RTC).

6.1.4 Reimplement Carrier Phase Tracking

Carrier phase tracking was removed from OSGPS to simplify porting. This should be reimplemented, along with hooks for future carrier phase-based positioning.

6.1.5 Faster Cold Acquisition

Cold acquisition time — the time from turn on to position output with no knowledge of position or time — can be significantly decreased. Once a satellite is acquired and identified, then its orbital slot indicates which other satellites in its orbit might possibly be in view, and which are beyond the horizon and can be ignored.

Another way to enhance cold acquisition time is to skip search bins, counting on the pull-in algorithm to pick up smaller changes in the correlator values than a direct match; this requires careful coordination with the tracking algorithm.

Finally, the obvious acquisition speed enhancement is to include the flash drivers in eCos and storing the almanac data in flash on every successful almanac acquisition. This is more difficult than it sounds because the flash must be erased in sectors, and it's likely that a full sector might not be available. Thus flash will

need to be copied to SRAM, and then written back along with the almanac. Note that the almanac is only useful if the real time clock on the GP4020 is backed up by battery (MG5001) or super-capacitor (SuperStar II).

6.1.6 Move Tracking Loops to Internal SRAM

To provide more memory and faster operation, all interrupt and tracking loop code should be moved to the zero wait-state 32-bit internal SRAM in the GP4020. This involves changing linker commands for RedBoot and eCos, as well as special compiler directives.

6.1.7 Code Refactoring and Contribution back to OSGPS

The OSGPS code base needs refactoring to make it more efficient and more clear. This is a long process of stepping through the code and applying comments, code style, and algorithmic refactoring as necessary. Once these changes are implemented and tested in GPL-GPS, these changes will be contributed back to OSGPS. There is even some possibility that OSGPS might move to using POSIX compliant threads, meaning that in many instances the exact same code could be operating on both GPL-GPS and OSGPS.

Some refactoring includes moving to equivalent but clearer algorithms. For example, moving from dithered correlators to tracking (early-late) correlators for the pull-in and tracking functions. While this will not fundamentally change the tracking algorithm, it is the standard way of explaining tracking in the literature and, thus, is easier to understand. Similarly, having the tracking loops maximize the in-phase (I) component versus the quadrature (Q) component is more of a

standard standard signal processing algorithm than maximizing Q.

6.1.8 Better Phase Locked Loop Algorithm

Research and implement a quasi-coherent PLL to replace the hybrid PLL currently used by OSGPS [15].

6.1.9 Network API (Flight Computer Model)

Create a network Application Programming Interface (API) to get correlator, high-rate (≥ 10 Hz) pseudorange, and/or navigation data to an offboard PC or “flight computer” and get aiding and initial position, velocity and time information back. It’s not clear how easily this can be implemented; the only options so far are a high speed serial bus (a 5 Mbps SPI synchronous serial port) or a direct 16 bit parallel interface which may not be possible with the MG5001 receiver board’s limited 51 pin connector.

6.1.10 Lock Aiding

Allow external sensor data to aid correlator tracking of satellites. For example, altimeter data can be used to allow fixes on only three satellites, and inertial data can be used to aid tracking of the satellite signals under high dynamic conditions.

6.1.11 DGPS

Implement a module to make GPL-GPS into a DGPS base station: placing the receiver at a known position allows the receiver to calculate atmospheric biases to

each satellite. Transmitting this bias to a roving receiver allows that receiver to subtract the propagation errors and obtain sub-meter positioning accuracy.

6.1.12 Attitude

Implement communication amongst GPL-GPS receivers, each doing carrier phase tracking and solving the integer ambiguity problem. With lock and carrier tracking confirmation from each receiver, the parallel system should be able to turn carrier phase into a full attitude.

6.1.13 Moving towards Open Hardware

An exciting, if not distracting, future project is to build an open hardware GPS receiver based on Zarlink's Orion reference design for the GP2015, GP2021, and ARM60 processor. There are real advantages to having open hardware: easy access to the parallel 16 bit bus means enhanced interfaces to peripherals and other processors, and external memory can be sized appropriately. Also, dividing the board into a RF and digital sections (with connectors) may allow for using the board as a front end for software receivers as well.

6.2 Conclusion

By porting open source software to commercial, off-the-shelf GPS receivers, GPL-GPS provides an accessible GPS receiver development environment that is otherwise unavailable. Its inexpensive and open nature lowers the barriers of entry to GPS development, which opens a rich variety of new and interesting GPS applications. Although just in its infancy, GPL-GPS is already being slated for

projects such as nanosatellites, UAVs, and small scale launch vehicles. The hope is that these GPL-GPS applications will then contribute their code changes back to the project, enabling GPL-GPS to quickly grow in maturity and features. Furthermore, GPL-GPS may enable academic research and hands-on coursework that was not previously possible. Now even a modest research grant can provide dozens of GPS development kits.

The vision for GPL-GPS is to lower the barrier to GPS development and enable a rich and vibrant community of GPS developers. I look forward to leading GPL-GPS through its first steps toward this vision.

Bibliography

- [1] Accelerated Technology, Inc. Nucleus homepage. http://www.acceleratedtechnology.com/embedded/nuc_rtos.html, May 2005.
- [2] Amateur Radio Satellite Corporation. Project Phase 3D. <http://www.amsat.org/>, May 2005.
- [3] Justin Armstrong. Jaysos 0.2 & waba vm for the gameboy advance. <http://www.badpint.org/jaysos/>, May 2005.
- [4] CadSoft Software. Eagle cad homepage. <http://www.cadsoft.de/>, May 2005.
- [5] D. Jeff Dionne and Michael Durrant. μ Clinux homepage. <http://www.uclinux.org/>, May 2005.
- [6] Takuji Ebinuma. ARMGPS homepage. <http://www.geocities.com/tebinuma/osgps/index.html>, May 2005.
- [7] eCos Community. eCos homepage. <http://ecos.sourceware.org/>, May 2005.
- [8] Free Software Foundation. GNU General Public License. <http://www.gnu.org/licenses/gpl.html>, May 2005.

- [9] Clifford Kelley, Joel Barnes, and Jingron Chen. 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.
- [10] Anthony J. Massa. *Embedded Software Development with eCos*. Prentice Hall, Upper Saddle River, NJ, 2003.
- [11] Wilhem Meignan. Isos real time operating system. <http://wilhem.meignan.free.fr/>, May 2005.
- [12] Micrium, Inc. μ C/OS-II product specification brochure. <http://www.ucos-ii.com/contents/products/uc-os-ii-RTOS.html>, May 2005.
- [13] Navman NZ Ltd. Navman oem solutions. http://www.navman.com/oem/products/gps_receivers/index.html, May 2005.
- [14] Nemerix, Inc. NJ1030 preliminary specifications. <http://www.nemerix.com/site/products/index.htm>, May 2005.
- [15] Bradford W. Parkinson and Jr. James J. Spilker, editors. *Global Positioning System: theory and Applications: Volume I*. American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 1996.
- [16] Portland State Aerospace Society. Homepage and project LV1b pages. <http://psas.pdx.edu/>, May 2005.
- [17] SiRF Technology, Inc. SirfStar ii/LP product brochure. <http://www.sirf.com/products-ss2eLP.html>, May 2005.
- [18] Gilbert Strang and Kai Borre. *Linear Algebra, Geodesy, and GPS*. Wellesley-Cambridge Press, Wellesley, MA, 1997.

- [19] Thales Navigation, Inc. Chipset comparison image. <http://products.thalesnavigation.com/en/solutions/oem/chipset.asp>, May 2005.
- [20] Trimble Navigation Limited. Trimble FirstGPS chipset. <http://www.trimble.com/firstgps.html>, May 2005.
- [21] James Bao-Yen Tsui. *Fundamentals of Global Positioning System Receivers*. John Wiley & Sons, Inc., New York, NY, 2000.
- [22] u-Nav Microelectronics, Inc. Baseband ics. http://www.unav-micro.com/baseband_ic.htm, May 2005.
- [23] Zarlink Semiconductor. *GP2021 GPS 12-channel correlator(DS4077)*, April 2001.
- [24] Zarlink Semiconductor. *GP2021 GPS receiver RF Front End (DS4374)*, February 2002.
- [25] Zarlink Semiconductor. *GP4020 GPS Baseband Processor Design Manual (DM5280)*, January 2002.

Appendix A

Initial Results

A.1 Introduction

As of May 2nd, 2005, GPL-GPS successfully calculates position and clock bias if four or more satellites have valid pseudoranges and ephemerides. However, without atmospheric corrections, a more robust locking algorithm, carrier phase positioning, and position filtering, the output position is very rough. Range errors are on order 100's of meters with infrequent excursions to 1,000's of meters.

To put the GPL-GPS results in context, 15 hours worth of data were recorded from a SigTec MG5001 receiver running with its commercial software. Later, 10 hours worth of data were taken on a MG5001 running GPL-GPS (May 13, 2005 CVS image).

A Connexant (now Navman) "Jupiter" GPS receiver board [13] was run using the same antenna. The positions from the Jupiter board were averaged to choose an independent measurement of the antenna position. The reference antenna position, within a few meters, is 45.47030° latitude, -122.62490° longitude, and 30 m altitude. In Earth Centered, Earth Fixed (ECEF) coordinates, that position is (-2,415,600 m, -3,773,550 m, 4,524,190 m). ECEF is a non-inertial right-handed

reference frame with the origin at the center of the earth, the Z axis through the geodetic north pole, and the X axis through the zero meridian on the equator.

Unfortunately, the testing environment is far from ideal: Figure A.1 shows the large trees blocking off a large fraction of the southern sky and most likely causing severe multipath interference. Future tests should be run at a site with a clearer view of the sky and less possibility of multipath interference.



Figure A.1: Comparison testing setup: roof-mounted antenna, antenna splitter and two MG5001 receivers on development boards

A.2 SigTec OEM Software Positioning Results

The `$GPGPQ,XYZ,1` command was sent to the MG5001 running the SigTec OEM software to produce 1 Hz ECEF coordinate messages. Surprisingly, the initial 500 points (approximately) of the ECEF position were an average of 26.6 km off of the antenna reference position (Figure A.2).

After the first 500 points, the SigTec software settled down and produce a more sane output (Figure A.3). The mean of the SigTec data set, minus the first 500 points, was $(-2,415,601 \text{ m}, -3,773,553 \text{ m}, 4,524,189 \text{ m})$ which is 3.4 m away from the antenna reference position. The maximum deviation from the sample mean

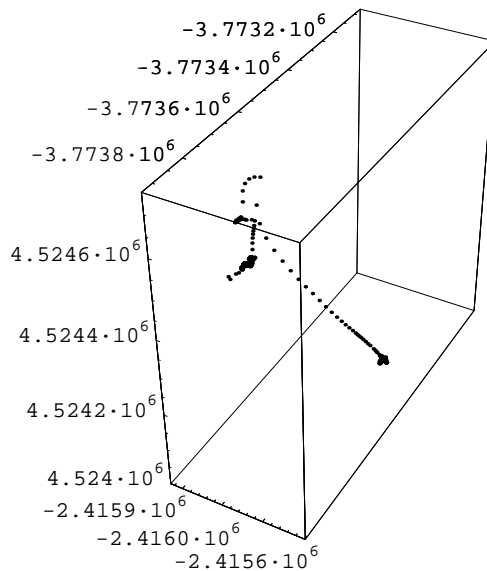


Figure A.2: Initial ECEF positions of the SigTec OEM software

was 94.3 m and the sample standard deviation was 7.9 m.

A histogram of the distance from each position to the data set average is shown in Figure A.4. One would think that the position set would be grouped near or on the mean, but the SigTec output seems to actively avoid the mean with a 12.8 m bias. A hypothesis to explain this behavior is that the output of the receiver tends to “clump”; an average of clumps might not necessarily fall in a clump, which would explain why there are almost no data points at the mean.

A.3 GPL-GPS Software Positioning Results

GPL-GPS was run for 10.1 hr to collect 14,225 points. Removing the farthest outliers, similar to the commercial software test, produced a data set of 14,049 points which represents a loss of 176 points (1.2% of the total sample). Since there

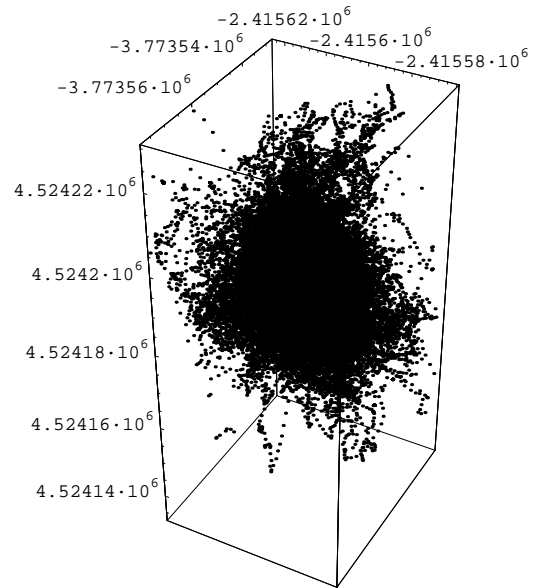


Figure A.3: 54,747 ECEF positions at 1 Hz from the SigTec OEM software

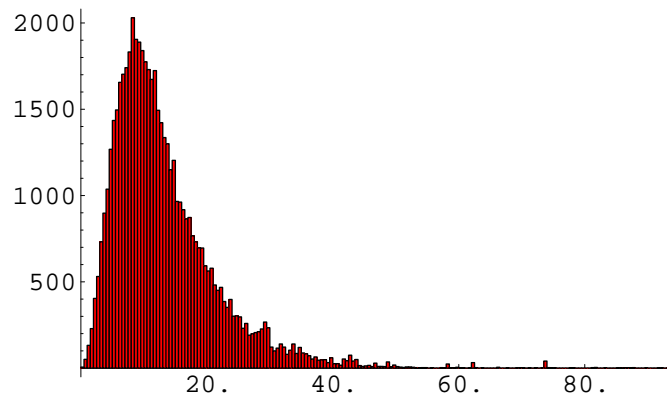


Figure A.4: Histogram of range to the sample mean for the SigTec OEM software

is no position filtering and the solution validity is not checked, some outliers are to be expected.

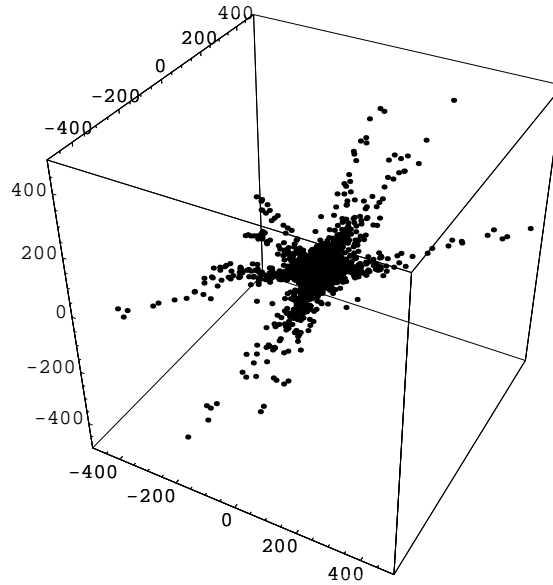


Figure A.5: GPL-GPS data bounded by a 1 km bounding box (98.8% of points)

The average of the data set was $(-9.1521 \text{ m}, 1.91896 \text{ m}, 5.08915 \text{ m})$, which is 10.7 m from the reference position. The mean of the data set from the average position is 41.7 m and the standard deviation is 80.3 m. Figure A.5 shows the data points with a 1 km bounding box. Note the trends (“offshoots”) in the data, most likely related to satellite geometry. Figure A.6 shows 90.5% of the data points falling in a 100 m bounding box. The 100 m box shows that the points are clustered in a sphere-like fashion around the mean, as expected.

Figure A.7 shows a histogram of the 1 km GPL-GPS dataset. Note the data centered around the 10 m range error.

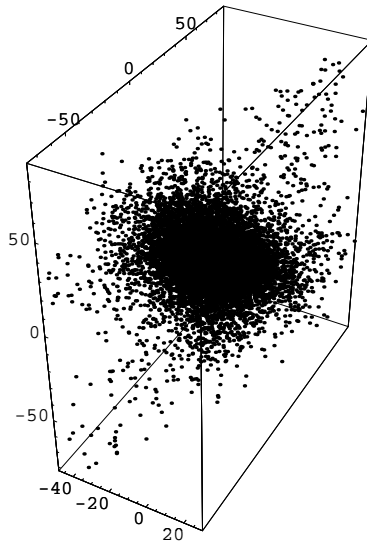


Figure A.6: GPL-GPS data bounded by a 100 m bounding box (90.5% of points)

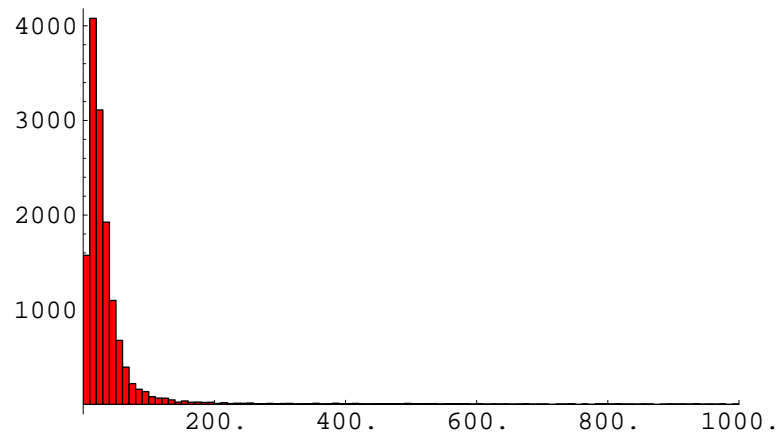


Figure A.7: Histogram of range to the sample mean for GPL-GPS (1 km bounded data set)

A.4 Receiver Time Series

Figure A.8 shows a time series of the distance to the sample mean from the SigTec software. Clearly, some kind of filtering is taking place since the adjacent points are highly correlated.

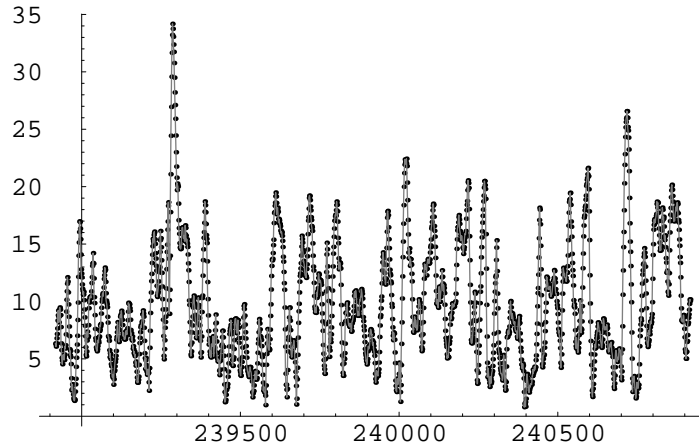


Figure A.8: Time series graph from the SigTec OEM Software

Conversely, the GPL-GPS time series of the distance to the sample mean (Figure A.9) shows no filtering. Note the large gap in points (the horizontal lines) indicate a time with less than four satellites in lock.

Figure A.10 shows a zoom in of the GPL-GPS time series.

A.5 Comparison Conclusion

While it is clear that the GPL-GPS positioning code needs further refinement, initial results are promising. Further improvements will be made with the addition of convergence checking, atmospheric modeling, carrier phase tracking, and

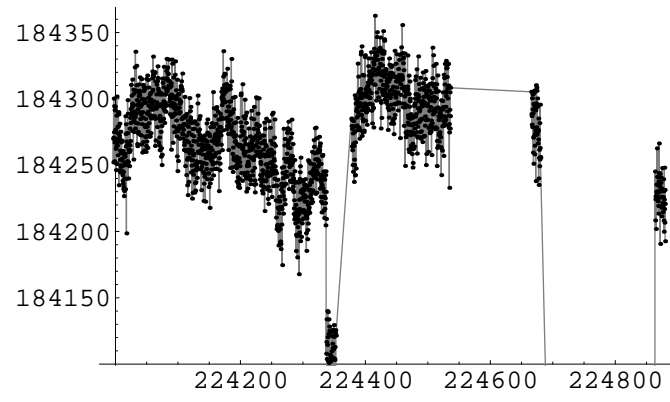


Figure A.9: Time series graph from GPL-GPS

position filtering.

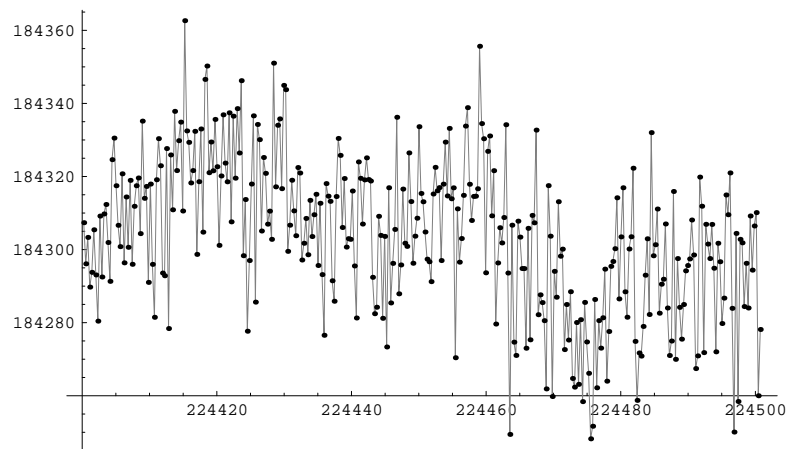


Figure A.10: Zoomed-in time series from GPL-GPS

Appendix B

The GPL-GPS Development Board

For complete printed circuit board design files, including bill of materials and PCB layout files, please see the GPL-GPS homepage at <http://gps.psas.pdx.edu/>. The schematic and PCB layouts were done in CadSoft's EAGLE CAD v4.1 [4], a freely available (but not open source) PCB CAD program for Linux, Macintosh and Windows.

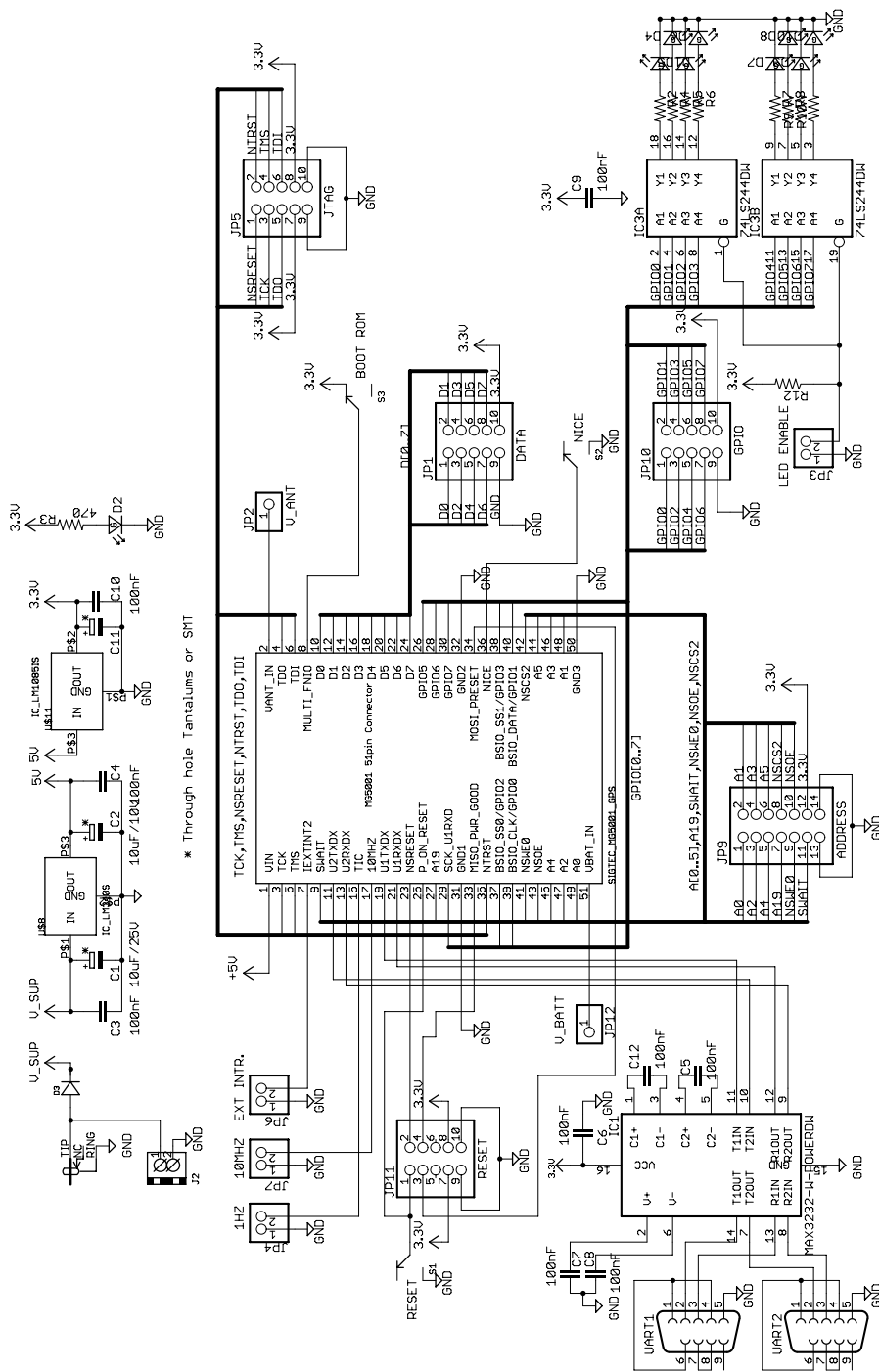


Figure B.1: Schematic of the GPL-GPS carrier board for the SigTec MG5001 receiver.