A Spaceborne GPS receiver

Benjamin J Nortier

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electronic Engineering with Computer Science at the University of Stellenbosch

Supervisor: Professor S. Mostert

December 2003

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

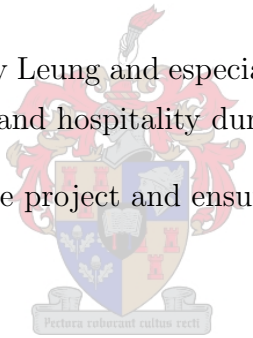
....

Signature Date

Acknowledgements

I would like to thank the following people for their contribution to the success of the project:

- My colleagues at the Electronic Systems Laboratory for advice and a great working atmosphere.
- Markus Markgraf and Sunny Leung and especially Dr. Oliver Möntenbruck at DLR for their expertise, support and hospitality during my stay.
- Prof. Mostert for driving the project and ensuring its success.



Abstract

The purpose of this study was to develop a Global Positioning System (GPS) receiver for use on a Low-Earth Orbit (LEO) satellite.

The study includes an examination of some of the fundamental GPS theory and how the LEO environment affects the operation of a GPS receiver. The hardware and software that was selected for the implementation are discussed. The reasons for porting the software to a new hardware platform and methods employed in the port are given. Thereafter the process of adapting the receiver software for use in space is given.

To verify the operation in space, the receiver was subjected to LEO simulations using a GPS signal simulator. These results are shown and discussed.

The tests indicated that the adaptations were successful and that the receiver will function on a LEO satellite.

Opsomming

Die doel van die tesis was om 'n Globale Posisionerings Stelsel (GPS) ontvanger to ontwikkel vir gebruik op 'n lae-wentelbaan satelliet.

Die studie begin met fundamentele GPS teorie en hoe die funksionering van die ontvanger beïnvloed word deur die wentelbaan van 'n satelliet. Die hardeware en sagteware vir die implementasie word bespreek. Die rede en metodes om die sagteware aan te pas om te werk op nuwe hardeware word gegee. Daarna word die proses om die sagteware aan te pas vir ruimtegebruik gegee.

Om the verifieer dat die ontvanger wel sal kan werk op 'n satelliet was dit getoets in 'n gesimuleerde ruimte-omgewing met 'n GPS seinsimulator. Hierdie resultate word gegee en bespreek.

Die toetse het gewys dat die aanpassings suksesvol was en dat die ontvanger in die ruimte sal funksioneer.

Contents

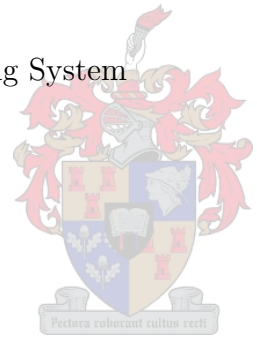
List of Figures

List of Tables

List of Acronyms

Acronym	Description
AGC	Automatic Gain Control
AXF	ARM Executable Format
bps	bits per second
BPSK	biphase shift key
C/A	Coarse/Acquisition
DOP	Dilution of Precision
ECI	Earth Centred Inertial
ECEF	Earth Centred Earth Fixed
ETX	End-Transmission
FLL	Frequency Lock Loop
GMST	Greenwich Mean Sidereal Time
GPS	Global Positioning System
IF	Intermediate Frequency
ISR	Interrupt Service Routine
JD	Julian date
kB	kilobytes (1024 bytes)
LEO	Low Earth Orbit
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
NORAD	North American Aerospace Defence Command
NPT	Nuclear Non-Proliferation Treaty

Acronym	Description
NVM	Non-volatile Memory
OBC	On-Board Computer
QPSK	Quadriphase shift key
PDOP	Position Dilution of Precision
PLL	Phase Lock Loop
ppm	Parts per million
PPS	Pulse-per-Second Synchronisation
PRN	Pseudo-random number
PVT	Position, Velocity and Time
RF	Radio Frequency
RAAN	Right Ascension of the Ascending node
RINEX	The Receiver Independent Exchange Format
RTC	Real-time clock
RTOS	Real-time Operating System
SGPS	Spaceborne GPS
SIGNAV	Sigtec Navigation
SEU	Single-Event Upset
STX	Start-Transmission
SV	Space Vehicle
US	University of Stellenbosch
TAI	International Atomic Time
TCXO	Temperature Compensated Crystal Oscillator
TCB	Task Control Block
TTFF	Time-to-First-Fix
TLE	Two-line element
TOW	Time-Of-Week
UT1	Universal Time
UTC	Universal Time Constant



Chapter 1

Introduction

1.1 Motivation

The Global Positioning System (GPS) is a satellite position system for which novel uses and applications are found regularly. In the past few years it has also found a place for use in space. Not only can GPS be used for calculating the position and velocity of a satellite, it can also for example be used for scientific experiments such as atmospheric studies and for relative navigation of satellite formations.

Spaceborne GPS (SGPS) is a concept that has been proven in space and a GPS receiver is becoming a standard subsystem in many Low-Earth Orbit (LEO) satellites. By using carrier phase measurements and multiple antennas, SGPS can be used for attitude determination. This can augment traditional attitude determination systems or even replace them completely.

The Electronic Systems Laboratory (ESL) at the University of Stellenbosch aims to be at the forefront of new and emerging satellite technologies such as SGPS. In future SGPS will become even more accurate when GPS satellites which emit the L2 Civilian Signal and L5 Civilian Signal come into operation in 2003 and 2005. Multiple-antenna receivers which utilise these signals can potentially replace other forms of attitude determination.

Thus, the aim was to develop SGPS skills and a modern receiver that could improve on the size, power consumption and processing power characteristics of existing SGPS receivers. An in-house SGPS solution would also yield the advantage of a receiver that can be adapted for specific mission requirements and/or scientific experiments.

1.2 Background

In 1991 the ESL was formed as part of the Electronic Engineering Department at the University of Stellenbosch (US). It would be a facility to develop and build SUNSAT and an environment for satellite research. SUNSAT was the first satellite of the University of Stellenbosch and it completed a successful mission after launch in 1999.

Developing an SGPS receiver builds on the success of the SUNSAT programme. The ESL in conjunction with Sun Space and Information Systems aims to continue doing research in satellite systems and their commercial applications.

This was the first entry of the US into the field of SGPS.

1.3 Literature Synopsis

Many good books regarding GPS in general exist. Two that have been found very useful are "Understanding GPS, Principles and Applications" ([7]) edited by Elliot D. Kaplan and "Global Positioning System: Theory and Applications" ([15]) edited by Bradford W Parkinson and James J Spliker Jr. Both these investigate the GPS system from fundamental theoretical concepts to implementation and the former is more understandable to someone new to GPS. The latter is a comprehensive work in two volumes that thoroughly studies the operation of the system and includes examination of many applications for GPS.

Concerning GPS for space use, the best sources of information are articles, papers and technical reports. The documents by Deutsches Zentrum fur Luft-und Raumfahrt - German Space Operations Centre (DLR-GSOC) and specifically by Dr. Möntenbruck are extremely insightful. These are available on the web at *http://www.weblab.dlr.de/rbrt/GpsNav/Orion/ReceiverDevelopment.html* and most are available to the public.

1.4 Objectives of this study

Overall, the object of the study was to develop a low cost SGPS receiver that can function on future satellites. The study also includes an examination of how a GPS receiver would

function in space as the GPS system was intended for terrestrial use. If the requirements are known, the adaptations needed to an existing receiver so that it can operate in space can be deduced. The final objective was to have a GPS receiver that can function in space, including testing results that would prove its functionality.

The structure of the thesis follows these aims:

- Chapter 2 is a study into the theory applicable to GPS and more specifically Space-borne GPS. This includes some fundamental concepts such as time, date and coordinate systems, operational modes of a GPS receiver and navigation algorithms.
- Chapter 3 examines the dynamic signal environment that an SGPS receiver needs to function in. Its aim is to quantify what the requirements of an SGPS receiver are and the visibility performance of a LEO receiver.
- Chapter 4 looks into the functionality of the tracking loops of a GPS receiver. It includes a proposed model to study the effect of the high dynamics on signal acquisition and lock.
- Chapter 5 discusses the hardware that was used to implement the SGPS receiver. This includes a description of the chipset and a comparison with the ORION receiver and then a proposed design for a 4-antenna receiver. The latter would enable SGPS attitude determination.
- Chapter 6 describes the software used for the receiver. It contains the motivation for and methods used to port the GPS Architect software from the previous ORION hardware to the Sigtec Navigation (SIGNAV) receiver.
- Chapter 7 lists the main software adaptations to the terrestrial software to enable LEO operation.
- Chapter 8 shows the results of testing the receiver during simulated LEO conditions. A GPS signal simulator was used to perform these tests. From the results space performance can be evaluated.
- Finally, Chapter 9 lists the conclusions of the thesis, together with recommendations for further study on this subject.

Chapter 2

GPS Fundamentals

2.1 Introduction

This chapter discusses the theoretical basis for the work. The most important topics that relate to SGPS will be discussed. They are:

- The different clock and date systems that are used in GPS, how they relate to each other (Section 2.2).
- The coordinate reference frames used in calculating the navigation solution and the orbit propagator and how to transform between the two. (Section 2.3).
- A discussion of the operating and startup modes of a GPS receiver and the circumstances that warrant each mode (Section 2.5).
- How the GPS constellation geometry in relation to the receiver affect the accuracy of the measurements, or Dilution of Precision (DOP) (Section 2.6).
- The data contained in the GPS navigation message and how this is required to calculate a position fix. (Section 2.4).
- The algorithms required to calculate the position of the receiver from the measured pseudoranges. (Section 2.7).

2.2 Time and Dates

2.2.1 TAI, UT1 and UTC time

Two bases for time are used:

1. The earth's rotation around its own axis and around the sun
2. Atomic time, which is derived from the SI second. The SI second was defined at the 13th official meeting of the International Committee of Weights and Measures in 1967 as "the duration of 9 192 631 770 periods of the radiation corresponding to the transition between two hyperfine levels of the ground state of the cesium 133 atom" [18]. From this the SI day is defined as 86400 seconds and the Julian century as 36 525 days.

There are three important time scales in GPS:

1. **International Atomic Time (TAI)**: A uniform time scale based on the atomic second. TAI is derived from an ensemble of atomic standards located throughout the world, which are statistically processed to calculate definitive TAI [10]
2. **Universal Time 1 (UT1)**: UT1 is based on the earth's rotation with respect to the Sun and is continually corrected for variations in orbital speed, inclination of the Earth's equator with respect to the orbital plane and polar motion. UT1 determines the actual orientation of the ECEF (Earth Centred Earth Fixed) coordinate system with respect to inertial space (see Section 2.3).
3. **Universal Coordinated Time (UTC)**: UTC is based on atomic time and UT1. UTC is also an atomic time scale, but leap seconds are added to UTC periodically to keep it in step with the Earth's rotation, or UT1. By international agreement, UTC is kept to within 0.9 s of UT1 [15]. From the start of the GPS calendar, 13 leap seconds have been introduced to UTC time at the dates in Table 2.1.

2.2.2 GPS Time

GPS is based on atomic standard time, similar to UTC. GPS time does not contain the integer leap seconds that are added to UTC time periodically, this would throw the GPS

Table 2.1: *UTC Leap Second Dates*

1981	JUL	1
1982	JUL	1
1983	JUL	1
1985	JUL	1
1988	JAN	1
1990	JAN	1
1991	JAN	1
1992	JUL	1
1993	JUL	1
1994	JUL	1
1996	JAN	1
1997	JUL	1
1999	JAN	1

P(Y)-code receivers out of lock. The GPS control segment attempts to keep GPS time to within 1 μs of UTC time [15].

GPS time is defined by GPS Week and Time-Of-Week (TOW) in seconds and one GPS week has 604800 seconds (7*24*3600). The GPS zero-time point is defined as midnight on the night of January 5 or 0:00 on January 6 1980 [4]. The GPS week starts at midnight on the Saturday of every week (January 5th was a Saturday). For example, GPS week 1201 starts on the morning of Sunday January 12th 2003 [3].

The navigation message contains parameters to convert GPS to UTC time, transmitted in the 24 MSBs (Most Significant Bits) of words 6–9 plus the 8 MSBs of word 10 in page 18 of Subframe 4. The precise algorithm defining the relationship between UTC and GPS time is given in [15, p.141]:

$$t_{UTC} = (t_e - \Delta t_{UTC})[modulo - 86400s] \tag{2.1}$$

Where t_{UTC} is in seconds and

Δt_{UTC}	=	$\Delta t_{LS} + A_0 + A_1(t_e - t_{ot} + 604800(WN - WN_t))$ [s]
t_e	=	GPS time as estimated by the user on the basis of correcting t_{SV} for factors given in Subframe 1 clock correction discussion as well as for ionospheric and SA (dither) effects
Δt_{LS}	=	delta time due to leap seconds
A_0 and A_1	=	constant and first-order terms of polynomial
t_{ot}	=	reference time for UTC data
WN	=	current week number (derived from Subframe 1)
WN_t	=	UTC reference week number

Disregarding the polynomial terms, GPS time differs from UTC by an integer amount of leap seconds (t_{LS}). To use the SGP4 orbit propagator to estimate receiver position (see Section 7.3), the coordinates need to be transformed from the ECI (Earth Centred Inertial) to ECEF reference frame by using the receiver time. Thus, this offset has to be removed because the coordinate GMST calculations are based on UTC time.

2.2.3 Julian Date

The Julian date (JD) is used in the algorithm for transforming from ECEF to ECI coordinates (sections 2.3 and 3.2). It is defined as a certain amount of days and fraction of days since a fundamental epoch, chosen as 12 Noon on January 1, 4713 BC. The Julian day is the number of days that have passed since this epoch. J2000.0 and GPS start is defined in Julian Date as follows:

J2000.0	=	JD 2 451 545.0	=	12:00 January 1 2000
GPS start	=	JD 2444244.5	=	00:00 January 6 1980

2.3 Coordinates

2.2.1 Earth Centred Inertial (ECI) Coordinate System

The ECI coordinate system is fixed in inertial space. The X-axis points to a fixed point in the heavens, the "First Point of Aries". The Z-axis coincides with the Earth's rotation vector, and the Y-axis completes the right-hand coordinate system. The Earth's equato-

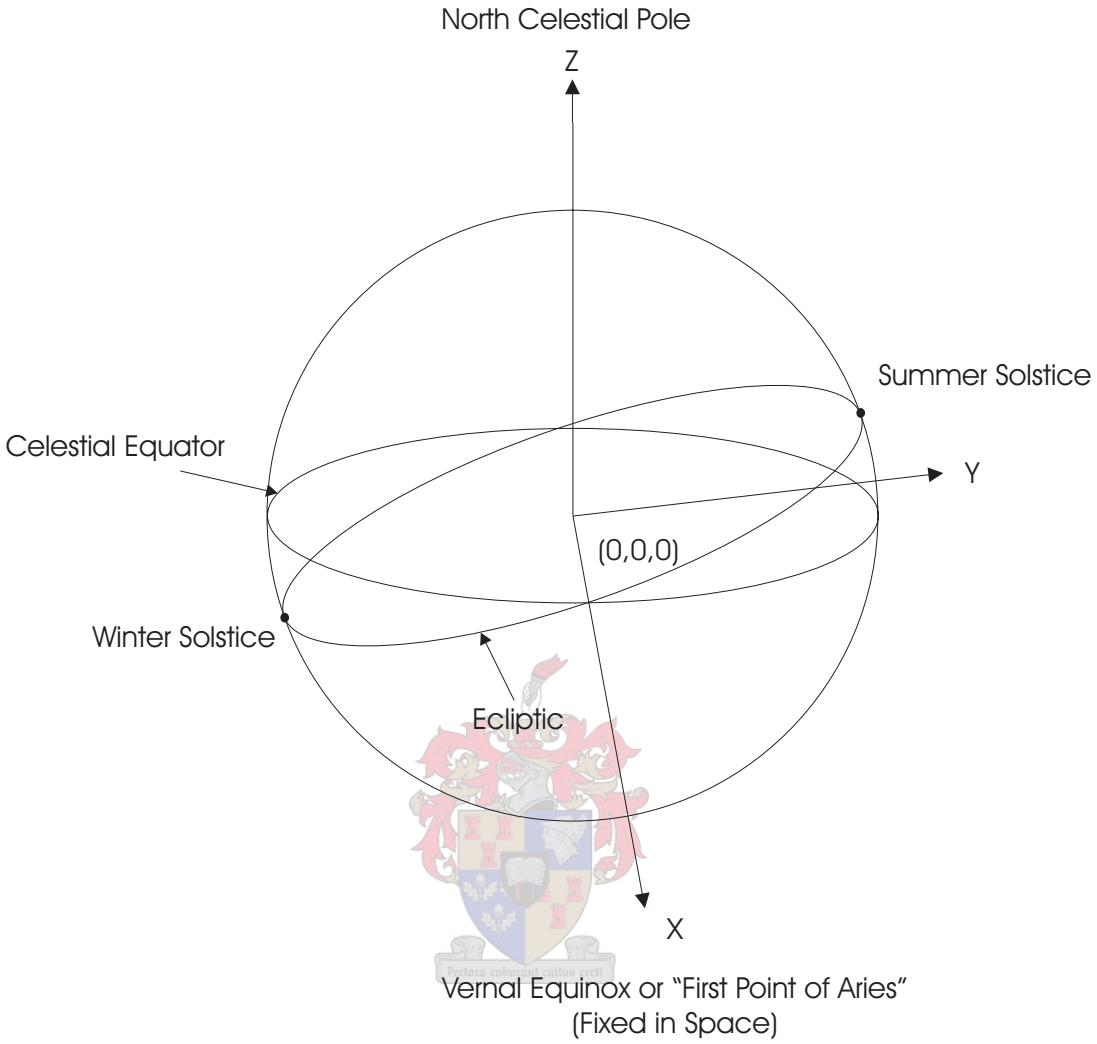


Figure 2.1: *ECI coordinate system*

rial plane coincides with the celestial equatorial plane, and the path of the Sun around the Earth is known as the ecliptic (see Figure 2.1).

2.3.2 Earth Centred Earth Fixed (ECEF) Coordinate System

The input and output of GPS broadcast data are in Earth-fixed WGS-84 (ECEF) system and in GPS time [1]. The ECEF coordinate system has its origin at the centre of the Earth (as defined in the WGS-84 coordinate system [24]) and the X-axis is the intersection of the Greenwich Meridian and the equatorial plane. The coordinate system rotates with the Earth's crust with an angular velocity of $7.2921151467 \times 10^{-5}$ rad/s relative to inertial space and the ECI coordinate system.

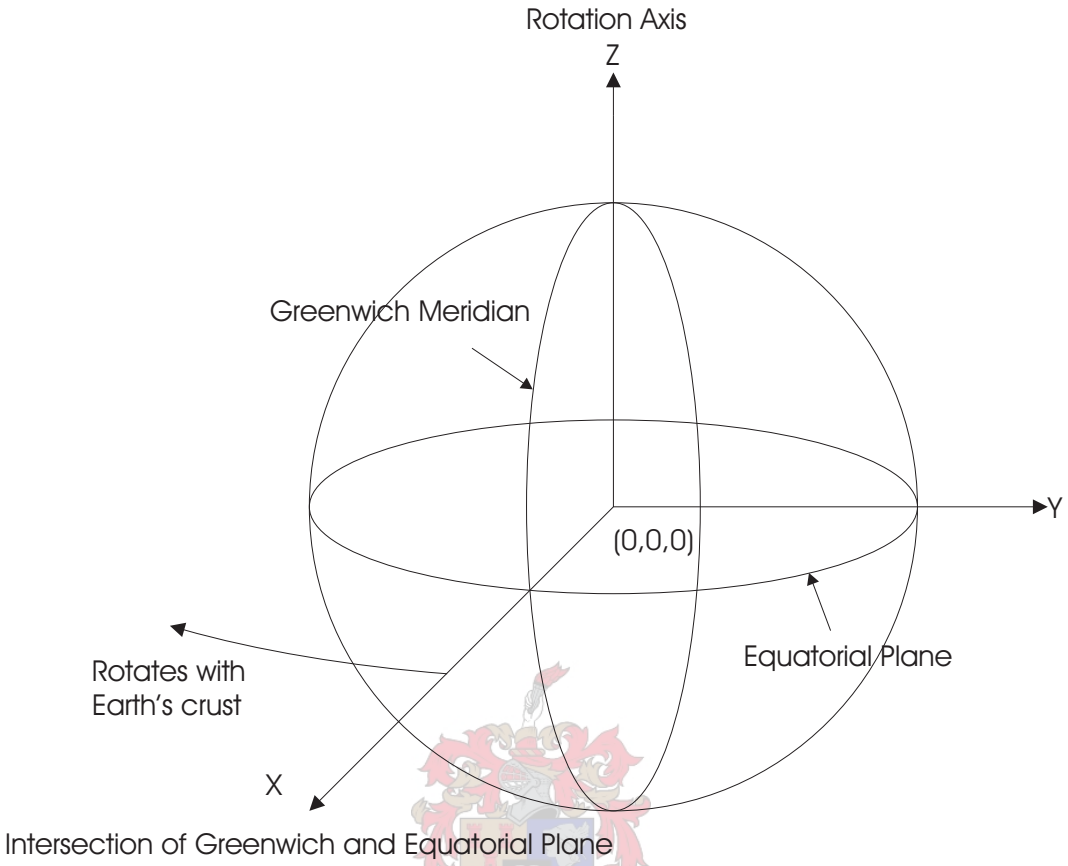


Figure 2.2: *ECEF coordinate system*

2.3.3 Coordinate System Transformation

The transformation from ECI to ECEF coordinates takes into account the rotation of the Greenwich prime meridian with respect to the space-fixed inertial system. Therefore this transformation relates the True of Date coordinates to the geographical coordinates corresponding to the rotating Earth [2]. The transformation is a rotation around the Z-axis of either coordinate system.

The transformation from ECI to ECEF coordinates for position and velocity [1] are:

$$\mathbf{r}_{ECEF} = \mathbf{R}_\theta \mathbf{r}_{ECI} \tag{2.2}$$

$$\dot{\mathbf{r}}_{ECEF} = \mathbf{R}_\theta \dot{\mathbf{r}}_{ECI} + \dot{\mathbf{R}}_\theta \mathbf{r}_{ECI} \tag{2.3}$$

with the sidereal rotation matrix $\mathbf{R}_\theta$ given by:

$$\mathbf{R}_\theta = \begin{bmatrix} +\cos\theta & +\sin\theta & 0 \\ -\sin\theta & +\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

and

$$\mathbf{R}_{\dot\theta} = \dot\theta \begin{bmatrix} -\sin\theta & +\cos\theta & 0 \\ -\cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.5}$$

The last term in the velocity equation (eq. 2.3) can be as expressed as:

$$\begin{aligned} \dot{\mathbf{R}}_\theta \mathbf{r}_{ECI} &= \dot\theta \begin{bmatrix} -\sin\theta x_{ECI} + \cos\theta y_{ECI} \\ -\cos\theta x_{ECI} - \sin\theta y_{ECI} \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{ECI}\dot\theta \\ -x_{ECI}\dot\theta \\ 0 \end{bmatrix} \\ &= \mathbf{R}_\theta \begin{bmatrix} x_{ECI} \\ y_{ECI} \\ z_{ECI} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \dot\theta \end{bmatrix} \\ &= \mathbf{R}_\theta(\mathbf{r}_{ECI} \times \omega) \end{aligned} \tag{2.6}$$

where $\omega^T = (0, 0, \dot\theta)$ and $\dot\theta = 7.2921151467 \times 10^{-5}$ rad/s, which is the Earth's rotation rate. Equation 2.3 reduces to:

$$\dot{\mathbf{r}}_{ECEF} = \mathbf{R}_\theta(\dot{\mathbf{r}}_{ECI} - \omega \times \mathbf{r}_{ECI}) \tag{2.7}$$

In the equations θ denotes the so-called Greenwich true (or apparent) sidereal time of the date, which is the angle between the first point of Aries and the intersection of the Greenwich meridian and equatorial plane. The Greenwich mean sidereal time (GMST) is given in [1]:

$$\theta = 1.0027379093\tau_{UTC} + \overline{\theta_0} + \Delta\varphi \cos\varepsilon' \tag{2.8}$$

where the first term on the right side accounts for the different scales of solar and sidereal time, $\Delta\varphi \cos\varepsilon'$ is known as equation of equinoxes, being a function of the nutation in longitude $\Delta\varphi$ and the true obliquity of the ecliptic ε'. The last term can be ignored for calculations close to J2000.0. τ_{UTC} is the amount of seconds since the start of the day (00:00 UTC). $\overline{\theta_0}$ is given by:

$$\overline{\theta_0} = 24110.54841 + 8640184.812866T + 0.093104T^2 - 6.2 \times 10^{-6}T^3 \tag{2.9}$$

where $\overline{\theta_0}$, in seconds, is the Greenwich mean sidereal time at 00:00 UTC of the day. T is the number of Julian centuries since J2000.0 to 00:00 UTC of the date in question.

For example, the Greenwich Mean Sidereal Time (GMST) at 14:00 UTC, 16 January 2003 is calculated as follows (remove multiples of 360 degrees from the answer):

Days since J2000.0	=	1111.083333333372
Days from J2000.0 to 0^h UTC	=	1110.5
T (No of Julian centuries)	=	1110.5/36525 = 0.03040383299110
$\overline{\theta_0}$	=	286805.2845586983 s
Fraction of day	=	0.58333333337214
τ_{UTC}	=	50400.00000335276 s
θ	=	337343.2751907802 s = 1405.596979961584°
$\Rightarrow GMST$	=	325.5969799615841°

2.4 Navigation Message

The GPS signal carries with it data that is needed by the receiver to obtain a position, velocity and time (PVT) solution. The GPS Interface Control Document [4] describes the exact format of the message. The data stream is transmitted at 50bps, is modulo-2 added to the C/A and P(Y) codes on the L1 frequency and may or may not be carried on the L2 frequency.

The navigation data are uploaded by the GPS Control Segment once per day or more often if required. The data stream is synchronous with the 1kHz C/A epochs. It is formatted into 30-bit words, the words are grouped into Subframes of 10 words each. Each Subframe is thus 6 seconds in duration. A frame consists of 5 Subframes and is 1500 bits in length and 30 s in duration. A Superframe is 25 frames with a duration of 12.5 minutes.

Much of the data in every frame is repeated, i.e. frames 1 to 3 have the same format, whereas frames 4 to 5 have 25 pages or different sets of data. Frames 4 and 5 contain the almanac data, which is used for Doppler prediction. The ephemeris data are repeated in each frame. Thus, to download the entire almanac from one SV (Space Vehicles), 12.5 minutes are required. To retrieve the ephemeris for one SV, a maximum of 30 seconds is needed. Every SV transmits only its own ephemeris data.

2.5 Operating Modes

The time taken for signal acquisition and Time-to-First-Fix (TTFF) for a GPS receiver depends on the GPS navigation data that is available (see Section 2.4) and the accuracy of the position estimation. Three startup modes are defined:

1. **Cold Start** - No Almanac or Ephemeris data are available for the GPS SVs : Typically during a Cold Start all the SV PRNs are searched progressively until lock is attained on one or more satellites. When signal lock is achieved, the Almanac and Ephemeris data can be downloaded. Downloading the Almanac from one satellite takes up to 12.5 minutes and Ephemeris data 30 seconds. Any GPS SV transmits only its own Ephemeris data, but the entire Almanac is transmitted.

2. **Warm Start** - Almanac data are available and it is used for signal acquisition: The Doppler frequency offsets are predicted by using the last known position fix and the current time (which may have to be programmed) together with the Almanac data. The Almanac contains coarse values that are used to estimate the positions and velocities of the GPS satellites and is valid for a maximum of 4 days. From signal acquisition on a specific SV, the Ephemeris data can be downloaded within 30 seconds. The Ephemeris data are required for a navigation fix.

3. **Hot Start** - Ephemeris and Almanac data are available: The Ephemeris data are used in the Doppler prediction process (this method is more accurate than using the Almanac). A navigation solution can be computed as soon as code and carrier lock is achieved since the Ephemeris data are available. Ephemeris data are valid for 4 hours.

The startup mode determines the way in which the correlator channels are assigned. During Cold Start the channels are assigned to SVs according to the search order. In

a Warm or Hot Start, the channels are assigned to satellites that the receiver predicts should be visible. The channels can e.g. be assigned to try and acquire the satellites with the highest elevations (Highest Elevation Mode), or satellites which yield a solution with the smallest position dilution of precision (PDOP) depending on design requirements.

2.6 Dilution Of Precision (DOP)

The concept of Dilution of Precision (DOP) will be illustrated by means of an example [7]. In Figures 2.3 and 2.4 two cases are presented. In GPS navigation, a calculated range to each GPS SV (a pseudorange - see Section 2.7.1) and the calculated positions of those SVs is used to determine the position of the receiver. For ease of illustration, assume that the receiver and GPS clock are synchronised and that the user knows the time of transmission and positions of the two SVs (These values are normally determined in the navigation solution).

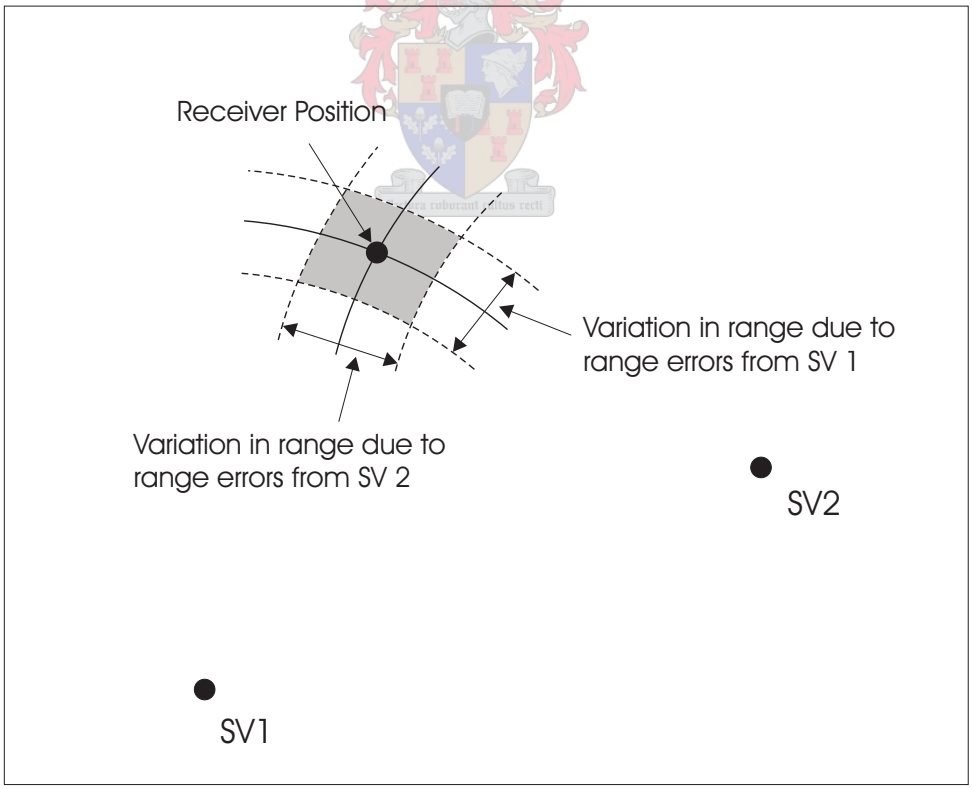


Figure 2.3: *DOP Geometry A*

DOP is the idea that navigation accuracy resulting from measurement error depends on the receiver/SV geometry. In case A (Figure 2.3) the angle formed between the receiver and the two SVs is an approximate right angle. In case B the angle is much smaller as

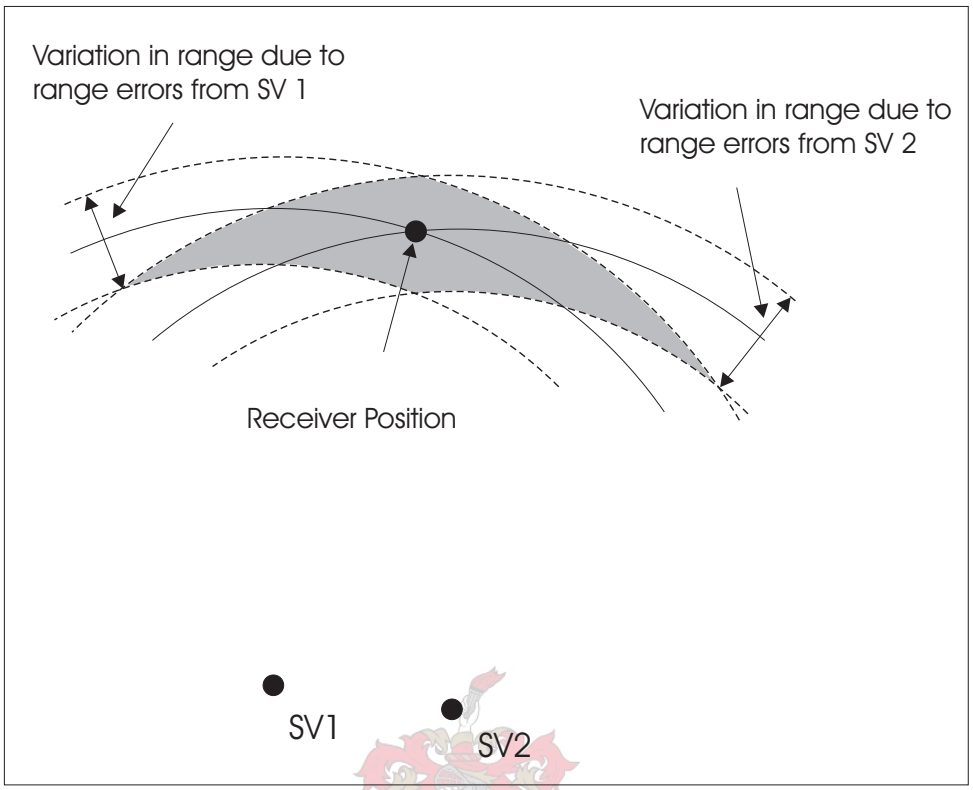


Figure 2.4: *DOP Geometry B*

the two GPS SVs are closer to each other.

The solid line denotes the true range ring from the SV to the receiver. The dashed line indicates the variation in the range rings resulting from ranging errors. For both examples the error is the same. The true receiver position is the intersection of the true range rings. However, the shaded area is the locations that are computed from the real measurements, which include some pseudorange error.

From the figures it is apparent that in example B, the accuracy of the navigation solution is much worse than in case A. Case B has a larger dilution of precision than case A. Thus when satellites are selected to use in the navigation solution, the current constellation geometry could be important.

In practice, there are normally more tracking channels (most commercial receivers have 12 tracking channels) available than there are visible satellites (See Section 3.4.2 for visibility studies). Consequently, satellite selection is usually done by selecting those with the highest elevation relative to receiver position. Also, the lower elevation satellites' measurement accuracy is worst affected by the atmosphere. If a situation arises where there are less tracking channels than visible satellites (See Section 5.3.3), DOP becomes

a criteria by which channels should be assigned.

2.7 Receiver Position and Velocity Determination

2.7.1 Position and clock offset solution

For a 3-dimensional PVT solution, at least 4 satellites are required. If 4 pseudoranges are available, the four unknown variables, (x_u,y_u,z_u) and t_u can be solved where (x_u,y_u,z_u) is the cartesian position and t_u the clock offset. If 4 pseudorange rates are available, (x_v,y_v,z_v) and t_v can be solved where (x_v,y_v,z_v) is the cartesian velocity and v_t the clock drift. An abbreviated derivation of the formulas for a position and time fix will be discussed and the full derivations can be found in [7, p.43].

A pseudorange measurement is a range measurement calculated from signal transit time. Thus, the measured pseudorange should be equal to the distance between the receiver and the satellites, plus the clock offset (difference between GPS and receiver time) times the speed of light:

$$\rho_j = \|\mathbf{s}_j - \mathbf{u}\| + ct_u \tag{2.10}$$

where $\mathbf{s}_j$ is the position of satellites 1 to 4 and c the speed of light. Expanding yields the following set of equations:

$$\rho_1 = \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} + ct_u \tag{2.11}$$
$$\rho_2 = \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} + ct_u \tag{2.12}$$
$$\rho_3 = \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} + ct_u \tag{2.13}$$
$$\rho_4 = \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} + ct_u \tag{2.14}$$

This set of non-linear equations can be solved by a closed-form solution, iteration or Kalman filtering. The iterative method will be illustrated, which is the method implemented in the GPS Architect software. When calculating a solution, the estimate of the receiver position $(\hat{x}, \hat{y}, \hat{z})$ is used as a point to linearise around. The true position is thus the estimate plus an offset $(\Delta x_u, \Delta y_u, \Delta z_u)$:

$$x_u = \hat{x}_u + \Delta x_u \tag{2.15}$$

$$y_u = \hat{y}_u + \Delta y_u \tag{2.16}$$
$$z_u = \hat{z}_u + \Delta z_u \tag{2.17}$$
$$t_u = \hat{t}_u + \Delta t_u \tag{2.18}$$

If a single pseudorange is

$$\rho_j = \sqrt{(x_j - x_u)^2 + (y_j - y_u)^2 + (z_j - z_u)^2} + ct_u = f(x_u, y_u, z_u, t_u) \tag{2.19}$$

then an approximate pseudorange is

$$\hat{\rho}_j = \sqrt{(x_j - \hat{x}_u)^2 + (y_j - \hat{y}_u)^2 + (z_j - \hat{z}_u)^2} + ct_u = f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u) \tag{2.20}$$

and therefore

$$f(x_u, y_u, z_u, t_u) = f(\hat{x}_u + \Delta x_u, \hat{y}_u + \Delta y_u, \hat{z}_u + \Delta z_u, \hat{t}_u + \Delta t_u) \tag{2.21}$$

Then this function is expanded about the approximate point and associated predicted receiver clock offset $(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u)$ using a Taylor series:

$$\begin{aligned} &f(\hat{x}_u + \Delta x_u, \hat{y}_u + \Delta y_u, \hat{z}_u + \Delta z_u, \hat{t}_u + \Delta t_u) = f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u) + \\ &\frac{\partial f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u)}{\partial \hat{x}_u}\Delta x_u + \frac{\partial f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u)}{\partial \hat{y}_u}\Delta y_u + \frac{\partial f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u)}{\partial \hat{z}_u}\Delta z_u + \\ &\frac{\partial f(\hat{x}_u, \hat{y}_u, \hat{z}_u, \hat{t}_u)}{\partial \hat{t}_u}\Delta t_u + ... \end{aligned} \tag{2.22}$$

and by using eq. 2.20 this simplifies to:

$$\rho_j = \hat{\rho}_j + \frac{\partial \hat{\rho}_j}{\partial \hat{x}_u}\Delta x_u + \frac{\partial \hat{\rho}_j}{\partial \hat{y}_u}\Delta y_u + \frac{\partial \hat{\rho}_j}{\partial \hat{z}_u}\Delta z_u + \frac{\partial \hat{\rho}_j}{\partial \hat{t}_u}\Delta t_u + ... \tag{2.23}$$

The partial derivative of the approximate pseudorange (eq 2.20) with respect to $\hat{x}_u$ is:

$$\frac{\partial \hat{\rho}_j}{\partial \hat{x}_u} = \frac{1}{2} \times \frac{2(x_j - \hat{x}_u)}{\sqrt{(x_j - \hat{x}_u)^2 + (y_j - \hat{y}_u)^2 + (z_j - \hat{z}_u)^2}} \times -1 = -\frac{x_j - \hat{x}_u}{\hat{r}_j} \tag{2.24}$$

where

$$\hat{r}_j = \sqrt{(x_j - \hat{x}_u)^2 + (y_j - \hat{y}_u)^2 + (z_j - \hat{z}_u)^2}$$

and the full set of partial derivatives of eq 2.20 are

$$\frac{\partial \hat{\rho}_j}{\partial \hat{x}_u} = -\frac{x_j - \hat{x}_u}{\hat{r}_j} \tag{2.25}$$

$$\frac{\partial \hat{\rho}_j}{\partial \hat{y}_u} = -\frac{y_j - \hat{y}_u}{\hat{r}_j} \tag{2.26}$$

$$\frac{\partial \hat{\rho}_j}{\partial \hat{z}_u} = -\frac{z_j - \hat{z}_u}{\hat{r}_j} \tag{2.27}$$

$$\frac{\partial \hat{\rho}_j}{\partial \hat{t}_u} = c \tag{2.28}$$

Now simplifying by using only the first-order partial derivatives of the Taylor series (eq. 2.23) to obtain a linear equation:

$$\rho_j = \hat{\rho}_j - \frac{x_j - \hat{x}_u}{\hat{r}_j}\Delta x_u - \frac{y_j - \hat{y}_u}{\hat{r}_j}\Delta y_u - \frac{z_j - \hat{z}_u}{\hat{r}_j}\Delta z_u + c\Delta t_u \tag{2.29}$$

Rearranging to put the unknown terms on the right:

$$\hat{\rho}_j - \rho_j = \frac{x_j - \hat{x}_u}{\hat{r}_j}\Delta x_u + \frac{y_j - \hat{y}_u}{\hat{r}_j}\Delta y_u + \frac{z_j - \hat{z}_u}{\hat{r}_j}\Delta z_u - c\Delta t_u \tag{2.30}$$

The unit vector pointing from the approximate position, $\hat{\mathbf{u}}$ to the position of satellite j, $\mathbf{s}_j$ is defined as:

$$\mathbf{a}_j = \frac{\|\mathbf{s}_j - \hat{\mathbf{u}}\|}{|\mathbf{s}_j - \hat{\mathbf{u}}|}$$

and notice that the direction cosines which make up the unit vector (a_{xj}, a_{yj}, a_{az}) are

$$a_{xj} = \frac{x_j - \hat{x}_u}{\hat{r}_j}\Delta x_u \tag{2.31}$$

$$a_{yj} = \frac{y_j - \hat{y}_u}{\hat{r}_j}\Delta y_u \tag{2.32}$$

$$a_{zj} = \frac{z_j - \hat{z}_u}{\hat{r}_j}\Delta z_u \tag{2.33}$$

Equation 2.30 now reduces to

$$\Delta\rho_j = \hat{\rho}_j - \rho_j = a_{xj}\Delta x_u + a_{yj}\Delta y_u + a_{zj}\Delta z_u - c\Delta t_u \tag{2.34}$$

Finally a set of linear equations with four unknowns, Δx_u,Δx_u,Δx_u and Δt_u are obtained that can be solved if 4 pseudoranges are available:

$$\begin{aligned}
\Delta\rho_1 &= a_{x1}\Delta x_u + a_{y1}\Delta y_u + a_{z1}\Delta z_u - c\Delta t_u \\
\Delta\rho_2 &= a_{x2}\Delta x_u + a_{y2}\Delta y_u + a_{z2}\Delta z_u - c\Delta t_u \\
\Delta\rho_3 &= a_{x3}\Delta x_u + a_{y3}\Delta y_u + a_{z3}\Delta z_u - c\Delta t_u \\
\Delta\rho_4 &= a_{x4}\Delta x_u + a_{y4}\Delta y_u + a_{z4}\Delta z_u - c\Delta t_u
\end{aligned} \tag{2.35}$$

And expressed in matrix form:

$$\begin{bmatrix} \Delta\rho_1 \\ \Delta\rho_2 \\ \Delta\rho_3 \\ \Delta\rho_4 \end{bmatrix} = \begin{bmatrix} a_{x1} & a_{y1} & a_{z1} & 1 \\ a_{x2} & a_{y2} & a_{z2} & 1 \\ a_{x3} & a_{y3} & a_{z3} & 1 \\ a_{x4} & a_{y4} & a_{z4} & 1 \end{bmatrix} \begin{bmatrix} \Delta x_u \\ \Delta y_u \\ \Delta z_u \\ -c\Delta t_u \end{bmatrix}$$

or

$$\Delta\rho = \mathbf{H}\Delta\mathbf{x} \tag{2.36}$$

And to find a solution:

$$\Delta\mathbf{x} = \mathbf{H}^{-1}\Delta\rho \tag{2.37}$$

This method works well if the approximation is close to the true receiver position. If not, multiple iterations can be used, in each case using the new computed position as the estimated position. If more than four satellites are available, an overdetermined solution using least squares can be applied which will minimise the error. The least squares technique can be found in [7]

2.7.2 Velocity and clock drift solution

The formulas for the velocity and clock drift solution are derived in the same way, this time using the pseudorangerates. A pseudorangerate is the time derivative of the pseudorange and it shows how fast the range is changing, i.e. the velocity, and the rate at which the receiver clock is drifting. From [7] or derivation the solution matrix, $\mathbf{H}$, is the same for the velocity and the position solution when substituting the cartesian position variables with the velocity variables and the clock offset variable with the clock drift variable.

2.8 Conclusions

Although the GPS system requires strong scientific and mathematical knowledge, there are many sources of information on the subject ranging from easy to investigations into relativity theory. Any user can begin at a level that he/she is comfortable with and progress from there.

In the next chapter the dynamic environment of an SGPS receiver will be examined.

Chapter 3

Receiver Dynamics

3.1 Introduction

The biggest stumbling block to implement a Low-Earth-Orbit (LEO) GPS receiver is the extreme dynamics of the receiver compared to terrestrial (ground-based and low-altitude) applications. Normal terrestrial receivers cannot operate on a LEO satellite because the Doppler search spaces do not adequately provide for the large frequency shifts and shift rates. These shifts occur as a result of the high velocities associated with satellites. Another limitation on the operation of a GPS receiver in a LEO environment, is the International Traffic in Arms Regulations (ITAR) which place an altitude and velocity limit of 60000 feet and 1000 knots on commercial GPS receivers. These translate to roughly 18.3 km and 0.51 km/s whereas typical LEO receiver parameters are 700 km and 7 km/s respectively, which obviously exceed these limits.

The purpose of this chapter is to:

- Discuss the methods for determining the position of a satellite (Section 3.2).
- Give a quantitive examination of the effect of the high dynamic environment of a LEO receiver on the L1 signal. (Section 3.3).
- Show simulations of the signal dynamics to be expected on a LEO receiver and an examination into the GPS receiver capability on a geosynchronous receiver (Section 3.4).
- Summarise the signal dynamic effects for an SGPS receiver, and show how the predictability of a satellite's orbit can be used to aid receiver operation. (Section

3.5).

3.2 Satellite position estimation

3.2.1 Introduction

Two methods of orbit determination are used: one for simulating a LEO receiver's position and velocity and estimating a LEO receiver's position; and another is used for GPS Satellite position and velocity fixes. The first is the combination of "North American Aerospace Defence Command" (NORAD) 2-line element (TLE) sets and the SGP4 orbital model [5]. The second method is using the GPS ephemeris contained in the GPS navigation message. The receiver uses the second method to determine where the GPS satellites are when computing a navigation solution and is more accurate than a orbit propagator. The SGP4 method gives the satellite's coordinates in Earth Centred Inertial (ECI) coordinates, whereas the position calculated using the GPS ephemeris data is in Earth Centred Earth Fixed (ECEF) coordinates.

3.2.2 SGP4 Orbit Propagator

For a Spaceborne GPS receiver, the Almanac together with an orbit propagator (such as SGP4) is used to predict the Doppler frequencies for signal acquisition.

NORAD maintains general perturbation element sets on all resident space objects [5] known as the NORAD TLE sets and these are used globally for orbit determination. The SGP4 model is used to compute positions and velocities of satellites from these element sets. The TLE sets contain mean values, and a model is required to reconstruct the periodic perturbations of the satellite's orbit. The SGP4 model was developed by Ken Cranford in 1970 [9] and is used for near-Earth (orbit period less than 225 minutes) satellites. The SDP4 model is used for higher-altitude satellites (orbit period exceeding 225 minutes).

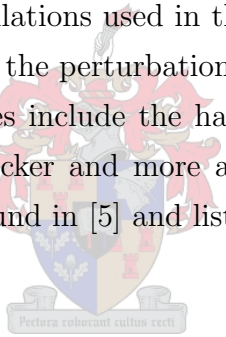
The SGP4 model is sufficient and flight-proven [11] for small satellite orbit prediction to aid GPS receiver operation. The onboard implementation requires only 8–10 kB of storage and can be implemented on a 80186-type processor [6]. The maximum accuracy of the SGP4 model, depending on the age of TLE data , is on the order of 2 km for position and 2 m/s for velocity, and for a frequency search bin bandwidth of 500 Hz (± 250 Hz)

a once-per-week TLE update strategy is sufficient [11]. The 2 m/s error in velocity is negligible considering that LEO velocities are around 7 km/s and that the velocity affects the Doppler shift of the carrier frequency. However, the errors increase significantly as the TLEs age, affecting the accuracy of the prediction. The SGP4 propagator is used for signal acquisition, so the TLE data that the receiver is using **need to be updated regularly**.

3.2.3 GPS SV

As discussed in Section 2.4, the navigation message contains values for calculating the GPS satellite positions.

The algorithm for calculating the satellite position using the ephemeris parameters is considerably simpler than the calculations used in the SGP4 model. This is because the TLE values are average values, and the perturbations have to be reconstructed using the SGP4 model. The ephemeris values include the harmonic correction terms and the fix calculated from these values is quicker and more accurate. The method to determine ECEF coordinates of each SV is found in [5] and listed in Appendix A.1.



3.3 The effect of LEO conditions on receiver operation

3.3.1 Doppler Shifts

As mentioned in the introduction, the receiver dynamics in a LEO environment is problematic for operation of the receiver. In terrestrial applications the Doppler frequency shifts are within ±5 kHz of the carrier [11] (L1 = 1575.42 MHz) whereas in space the shifts will be up to ±60 kHz (see Section 3.4) or more. Doppler frequency shift is calculated as follows:

$$f' = f_0\sqrt{\frac{1+v/c}{1-v/c}} = f_o + f'_d \tag{3.1}$$

thus

$$fd' = f_o \left[\sqrt{\frac{1 + v/c}{1 - v/c}} - 1 \right] \tag{3.2}$$

Where f_0 = Carrier Frequency (GPS L1 freq. = 1575.42 MHz)
f' = New frequency
fd' = Doppler shift
c = speed of light (299.792458×10^6 m/s)
v = relative receiver velocity

From [23, page 138] the velocity for a satellite in a circular orbit is:

$$V_{cir} = \sqrt{\mu / r} \tag{3.3}$$

Where $\mu \equiv GM$ = Earth's gravitational constant (398600.5 $\text{km}^3/\text{s}^{-2}$)
r = Orbit radius in km

Taking typical LEO and GPS altitudes of 700 and 20000 km respectively yield velocities in the order of 8 km/s and 4 km/s. Considering that because of the altitude difference the relative velocity will never reach ±12 km/s and consequently using a rough value of 10 km/s, the maximum Doppler shift in the carrier frequency will be in the order of ±55 kHz. As mentioned, in a terrestrial receiver the search space is only within a region of ±5kHz.

The Doppler shift does not only affect the carrier, but also the code frequency. The Doppler effect is inversely proportional to the wavelength of the signal and thus the code frequency shift is much less. Carrier aiding is used to adjust the code frequency as the shift is more apparent in the carrier frequency. The scale factor for the C/A signal on L1 between the carrier and code frequency is 1/1540 [7].

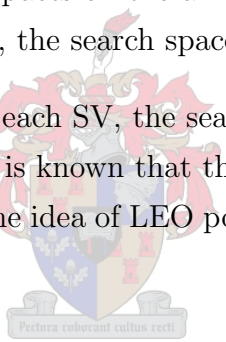
Not only is Doppler shift a factor to be considered, but the rate of Doppler shift could also be problematic. The carrier tracking loops in the receiver may be unable to maintain lock on the signal if the Doppler shift rate exceeds its design limits. If phase-lock loops (PLLs) are used for tracking, the Doppler shift rate becomes an issue since the PLLs bandwidth is much less than that of a frequency lock loop (FLL).

3.3.2 Frequency Search Bins

To understand why the Doppler shift plays an important role, a brief explanation of the signal search is required. The tracking loops in the receiver (carrier and code) have a certain bandwidth (typically 500 Hz). This means that a signal can be acquired if it is offset from the search frequency by ± 250 Hz. As explained in Section 3.3.1 the Doppler offsets for a terrestrial receiver can be as much as ± 5 kHz.

The loop bandwidth and possible frequency range determines that a range of frequencies have to be searched to obtain the GPS signals. Each frequency bin has to be searched for the code phase — the received signal has to be correlated with a known PRN code over the whole possible offset, which takes time. The search algorithm dwells in each frequency search bin to search all the code phases. This period is called the dwell time. The receiver oscillator error also impacts on the amount of frequency bins required for a search. If the possible error is large, the search space increases.

By predicting the Doppler offset of each SV, the search times can be reduced. The same is true for satellite visibility — if it is known that the SV is behind the earth time is not wasted by searching for its PRN. The idea of LEO position estimation is examined in 3.2.



3.3.3 Elevation Masks

When a GPS receiver operates on the ground or low altitude, the earth shadows any satellites that are below the horizon, and signals from GPS satellites at a very low elevation have to travel through much more atmosphere that those from satellites overhead. This implies large signal loss and increased signal delays which affect the fix accuracy.

However, when the receiver resides on a LEO satellite, the horizon of the earth is not at zero degrees relative to the satellite's local horizon anymore, but at a negative elevation, see Figure 3.1. It is thus possible to receive signals from satellites with a "negative" elevation.

The factors that determine the minimum elevation are the radius of the earth and the height of the troposphere. The troposphere causes significant delays and also bends the signal which degrade the calculation accuracy. Therefore, an elevation mask is chosen that minimises these effects, selecting the edge of the troposphere as a limit, not the earth's horizon. It will also be shown that a LEO satellite is within the -3dB beam width of the GPS signal.

Typical parameters that are used in the calculations are:

Tropospheric height: $h_{trop} = 12$ km
Earth Radius: $R_E = 6378$ km
GPS satellite height: h_{GPS} 20000 km
LEO satellite height: $h_{LEO} = 700$ km

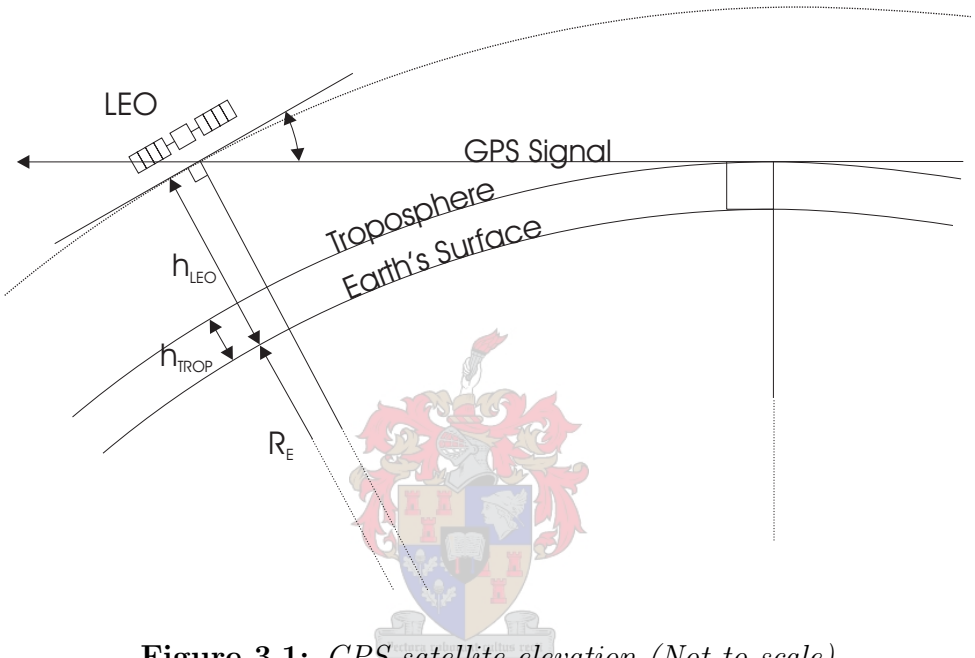


Figure 3.1: *GPS satellite elevation (Not to scale)*

For calculating the minimum elevation that LEO satellites can receive GPS signals, from Figure 3.1:

$$\sin(90^\circ - \phi) = \frac{R_E + h_{trop}}{R_E + h_{LEO}} \tag{3.4}$$

The minimum elevation angle is then calculated with the typical values from eq. 3.4 as

$$\phi = \cos^{-1} \frac{R_E + h_{trop}}{R_E + h_{LEO}} = 25.5^\circ \tag{3.5}$$

In terrestrial receivers an elevation mask of 5 degrees above the horizon is usually used. It would thus be important to also include satellites between -25 ° and +5° in the satellite selection software, **depending on receiver antenna design**. If the LEO satellite is always nadir pointing and the receiver antenna resides on the face always pointing away from earth (-Z axis), a elevation mask of +5 degrees may have to be used as only GPS SVs above the satellite's local horizon are visible.

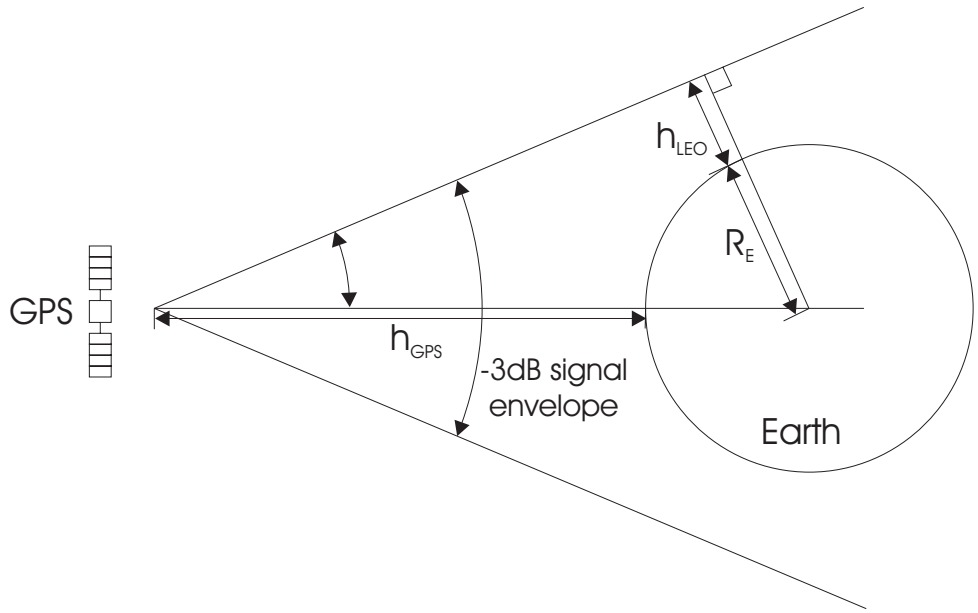


Figure 3.2: *GPS -3db signal Envelope (Not to scale)*

From Figure 3.2:

$$\sin\theta = \frac{R_E + h_{LEO}}{R_E + h_{GPS}} \tag{3.6}$$

Using the GPS signal envelope $\theta = 21.3°$ (see 3.4.3), the maximum height for a LEO satellite so that it still falls within the -3dB beam width is calculated as $h_{LEO} = (R_E + h_{GPS}) * \sin\theta - 6378 = 3203.8$ km. Thus all LEO satellites fall within the signal envelope.

3.4 Simulations

3.4.1 Max theoretical Doppler

The maximum theoretical Doppler shift will occur when the LEO and GPS satellites are in the exact same orbital plane (with different altitudes) and moving in opposite directions. Figure 3.3 shows the diagram used to determine the equations. Two bodies are moving toward each other at velocities v1 and v2 respectively and at time t=0 they are directly above each other, or on the y axis.

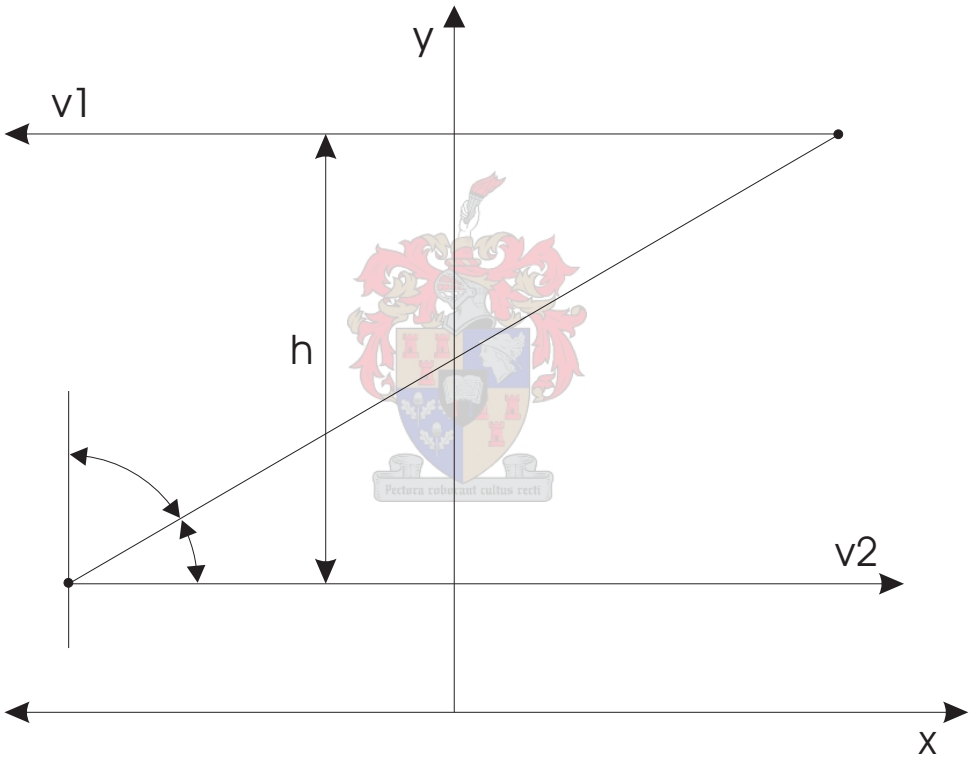


Figure 3.3: *Closing velocity diagram*

The angle formed by a line connecting the two bodies and a line parallel to the x axis is θ. The closing velocity between the two bodies is $\cos(\theta) \times (v_1 + v_2)$. Because arctan is defined for angles of between -90 and 90 degrees, ϕ is used instead of θ as θ varies between 0 and 180 as the two bodies move past each other and $\cos\theta = \sin\phi$. At any moment the x-distance between the bodies is $-t(v_1 + v_2)$ and the y-distance is h or the difference in altitude. Thus:

$$\phi = \tan^{-1} \frac{-t(v_1 + v_2)}{h} \tag{3.7}$$

$$v_{closing} = \sin\phi \times (v_1 + v_2) \tag{3.8}$$

Substituting eq. 3.7 into eq. 3.8 and the result into 3.2 and using MATLAB's symbolic toolbox an expression for the Doppler shift as a function of time is derived as follows:

```
h = sym('h'); v1 = sym('v1'); v2 = sym('v2'); c = sym('c');
v = sym('v'); t = sym('t'); f0 = sym('f0');

phi = atan(t*(v1-v2)/h);
v = sin(phi)*(v1-v2);
dopplershift = f0*(sqrt((1+v/c)/(1-v/c))-1);
ddoppler = diff(dopplershift);
```

And the resulting equations simplified by MATLAB's `simple(), simplify()` and `pretty()` are:

$$f'_d(t) = f_0 \left[\sqrt{\frac{1+A}{1-A}} - 1 \right] \tag{3.9}$$

$$\text{Where } A = \frac{-t(v_1+v_2)^2}{hc\sqrt{1+\left[\frac{t(-v_1-v_2)}{h}\right]^2}} \tag{3.10}$$

$$\frac{df'_d(t)}{dt} = \frac{-f_0(v_1+v_2)^2h^2c}{(Bc+tv_1^2+2tv_1v_2+tv_2^2)^{3/2}B\sqrt{Bc-tv_1^2-2tv_1v_2-tv_2^2}} \tag{3.11}$$

$$\text{Where } B = \sqrt{h^2+t^2v_1^2+2t^2v_1v_2+t^2v_2^2} \tag{3.12}$$

Figures 3.4 and 3.5 show these equations evaluated for a sample LEO satellite with a circular orbit at a height of 794 km (v1=7.5 km.s) and an GPS SV (v2=3.9 km/s) with difference in height of 19 394 km. These values yield a maximum Doppler shift of 58.5 kHz and maximum absolute shift rate of 34.8 Hz/s.

The Doppler frequency shift is zero when t = 0 (i.e when the relative velocity is 0). To obtain the maximum Doppler shift, the limit of the Doppler shift function (eq 3.9) is obtained:

$$f'_{d,max} = \left| \lim_{t\to\infty} f'_d(t) \right|$$

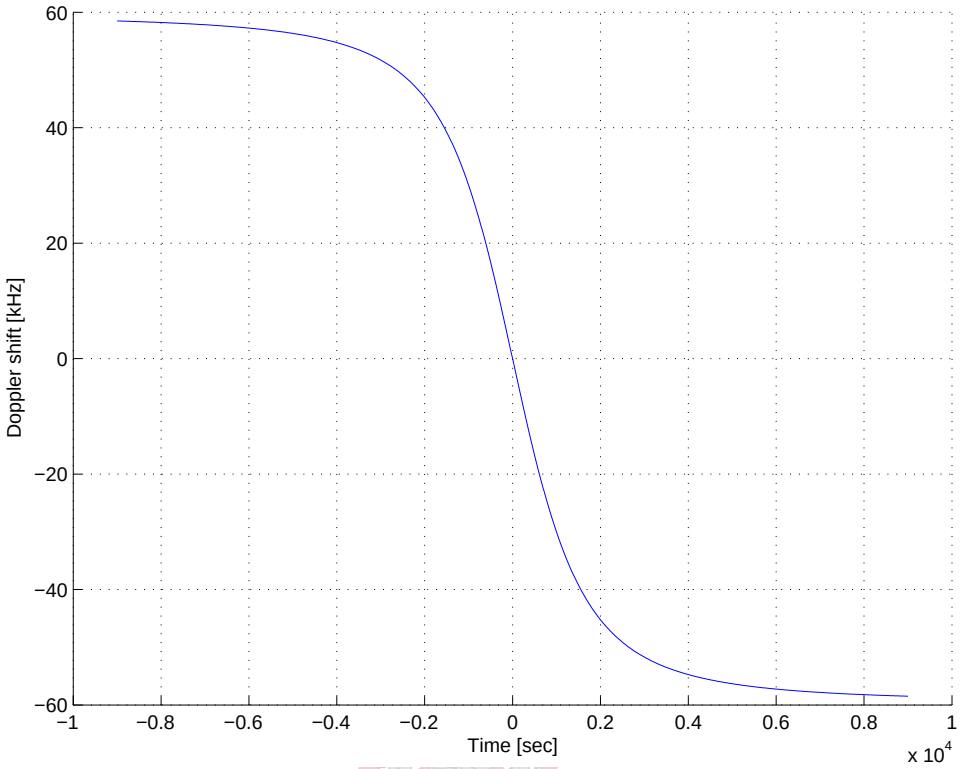


Figure 3.4: *PCSAT Doppler shift results*

Using MATLAB to evaluate this limit function at different LEO altitudes (assuming a circular orbit) the result in Figure 3.6 is obtained. The maximum Doppler shift rate occurs at t=0, so the maximum shift rate is obtained from eq 3.11 with t=0, v2 constant and v1 determined by the circular LEO satellite height. The result is plotted in Figure 3.7.

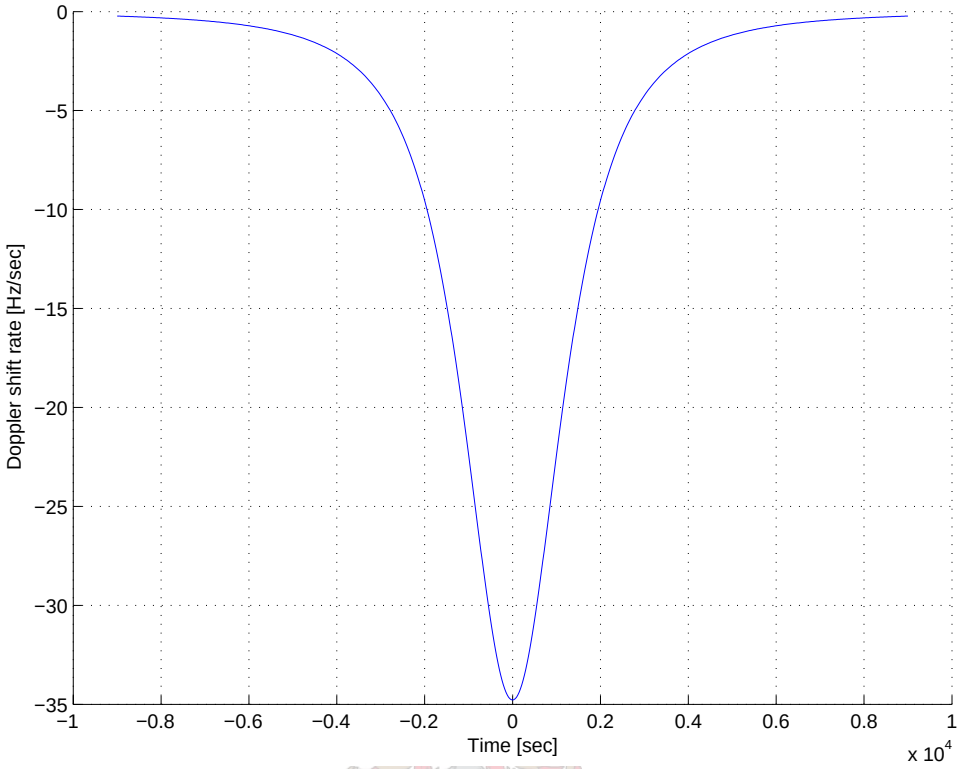


Figure 3.5: *PCSAT Doppler rate results*

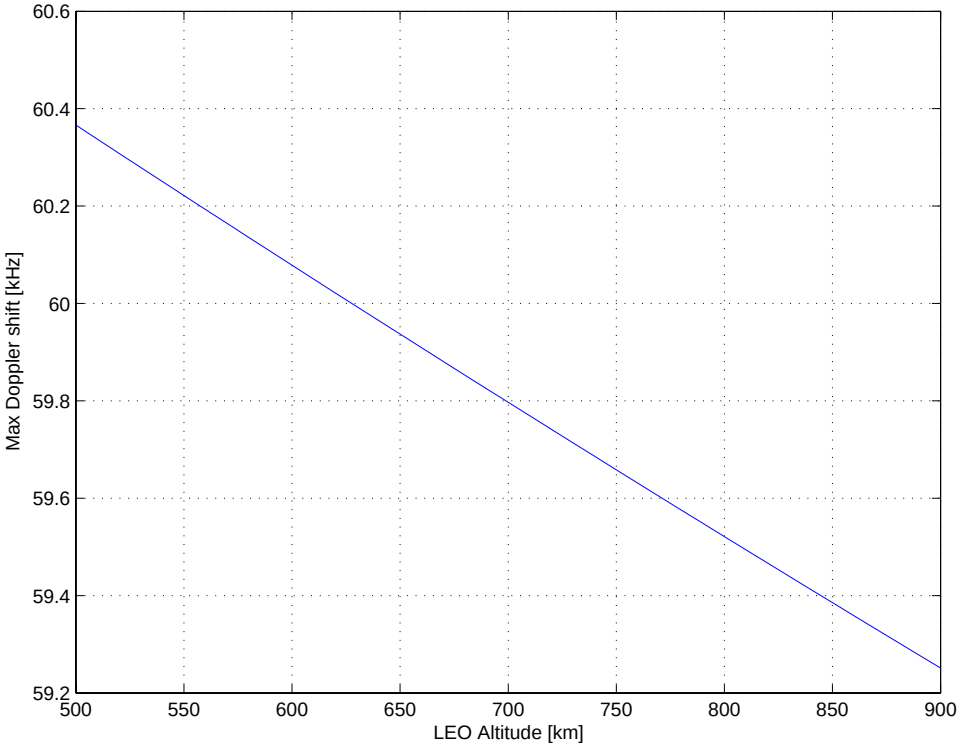


Figure 3.6: *Maximum Doppler shift vs. LEO altitude*

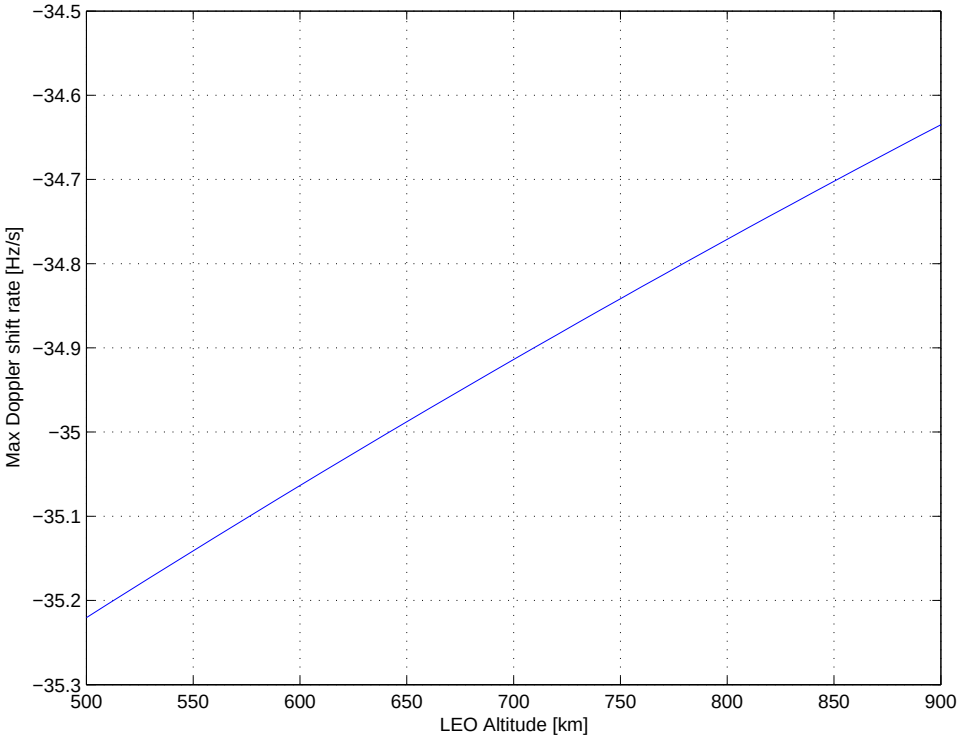


Figure 3.7: *Maximum Doppler shift rate vs. LEO altitude*

3.4.2 GPS constellation and sample LEO satellites

Simulations of the Doppler frequency shifts were done using MATLAB with sample LEO satellites over a period of 25 hours. Ephemeris data was used to calculate the GPS SV positions, and the TLE data together with the SGP4 propagator to calculate the position of the sample LEO satellite. The PCSAT, TUBSAT and SUNSAT satellites were used and the orbit parameters for these satellites are:

Name	Period [min]	Altitude [km]	Inclination [deg]
PCSAT	101	794	67
TUBSAT	99	730/719	98.4
SUNSAT	100	857/644	96.5

Simulations were done to determine:

- Number of visible GPS satellites for a sample LEO receiver using two elevation masks: 0 degrees and -25 degrees (see 3.3.3). Visibility results for PCSAT are shown in Figures 3.8 and 3.9. The results of visibility simulation of all the sample LEO satellites are listed in Table 3.1.
- Doppler shift of the sample satellites. The results of the simulation for Doppler shifts can be seen in Figures 3.10, 3.11 and 3.12.

Table 3.1: *GPS Satellite visibility simulation results*

LEO Satellite	Elevation Mask	Minimum	Average	Maximum
PCSAT	0°	6	9.9	14
	−25°	12	16.0	20
SUNSAT	0°	6	9.8	15
	−25°	11	16.2	21
TUBSAT	0°	6	9.9	14
	−25°	11	16.2	20

The results indicate that on average there are more than 12 visible satellites if a negative elevation mask is used. Depending on the receiver antenna design and satellite orientation there may not be enough correlator channels (usually 12) to track all the visible satellites. Receiver software may have to be modified to include a satellite selection algorithm to assign correlator channels according to minimum PDOP.

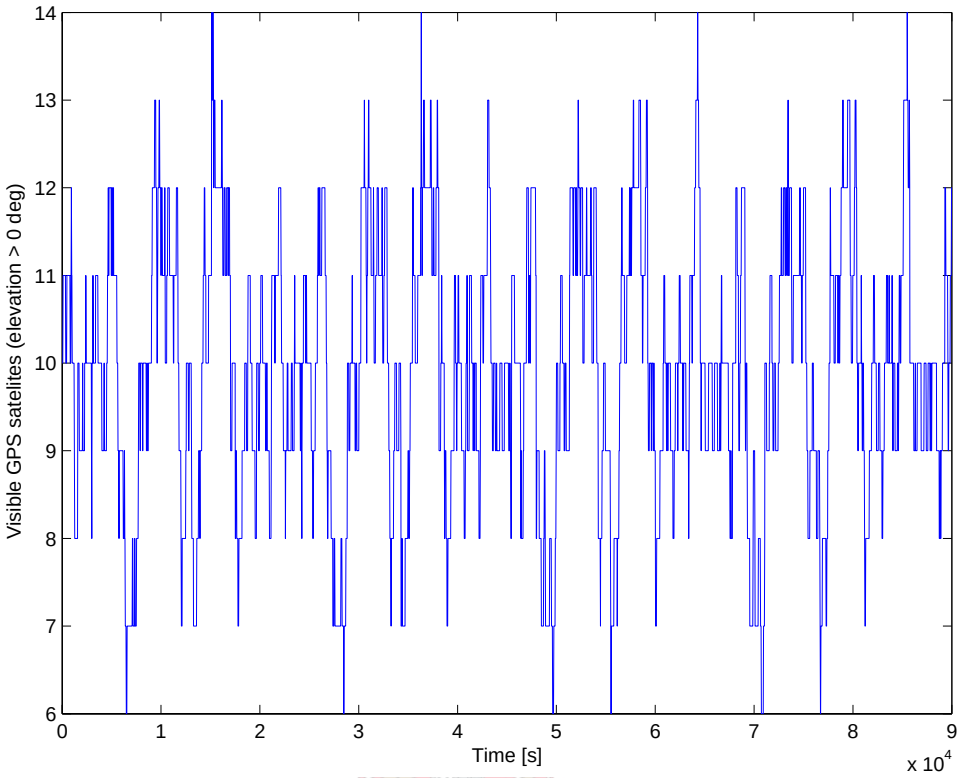


Figure 3.8: *PCSAT Visibility (0 deg elevation mask)*

The maximum Doppler shifts are in the order of 56 kHz. The individual maximum shifts are listed in Table 3.2. These results correspond with the approximated maximum closing velocity of 10 km/s calculated in 3.3.

Table 3.2: *Maximum Doppler Shift Results*

Name	Shift [kHz]
PCSAT	54.5735
TUBSAT	56.2455
SUN	56.8309

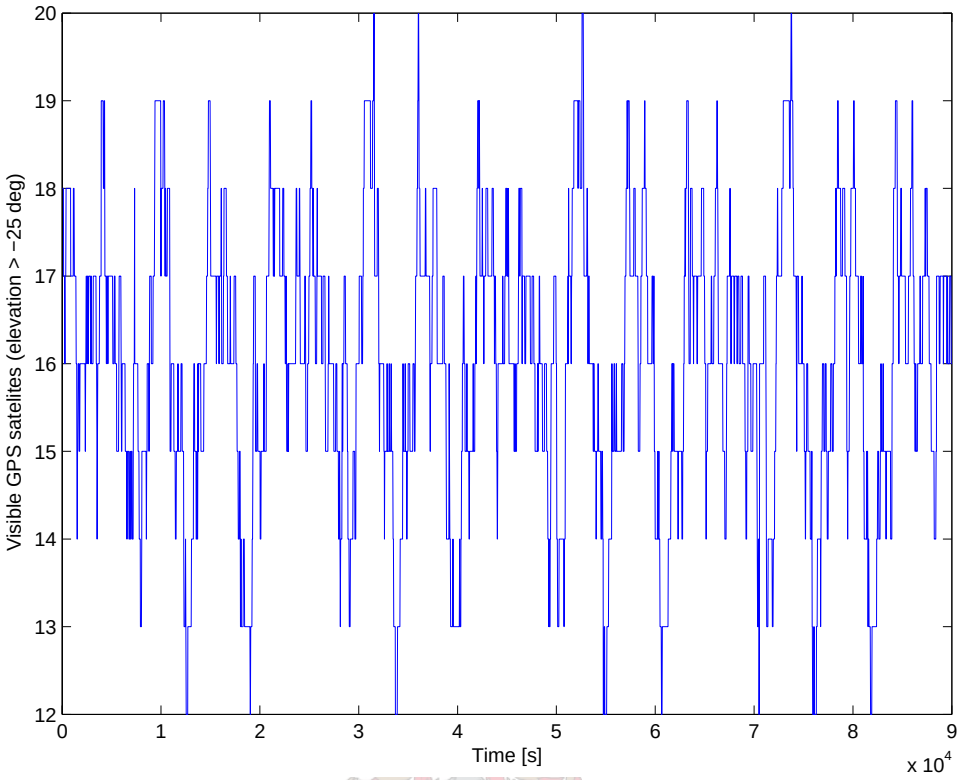


Figure 3.9: *PCSAT Visibility (-25 deg elevation mask)*

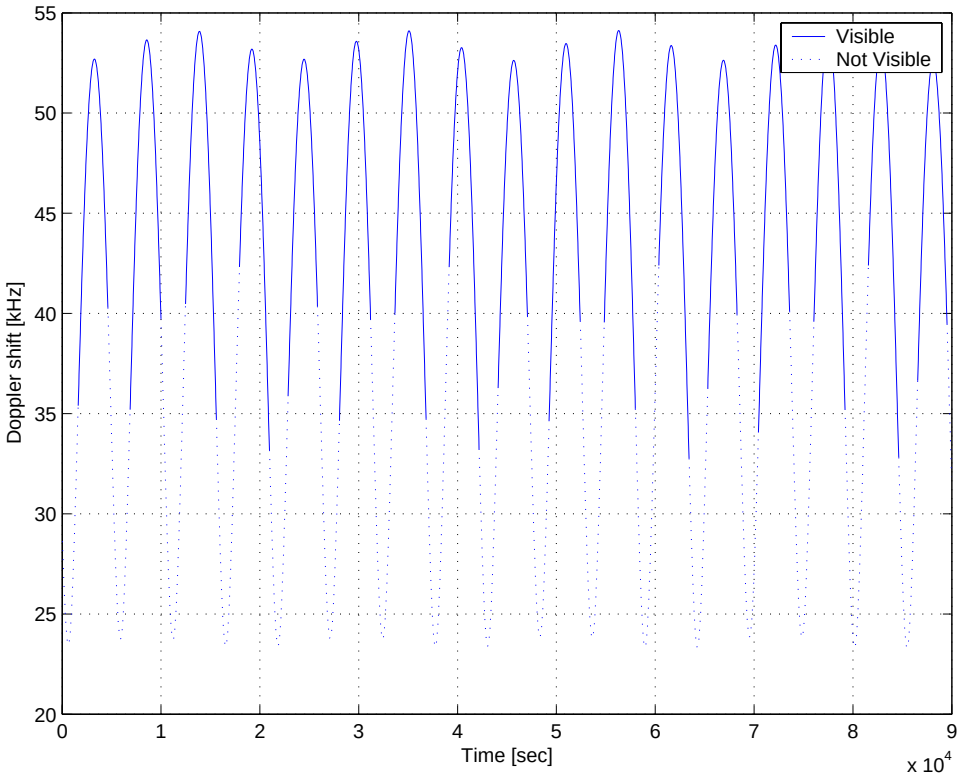


Figure 3.10: *PCSAT simulation*

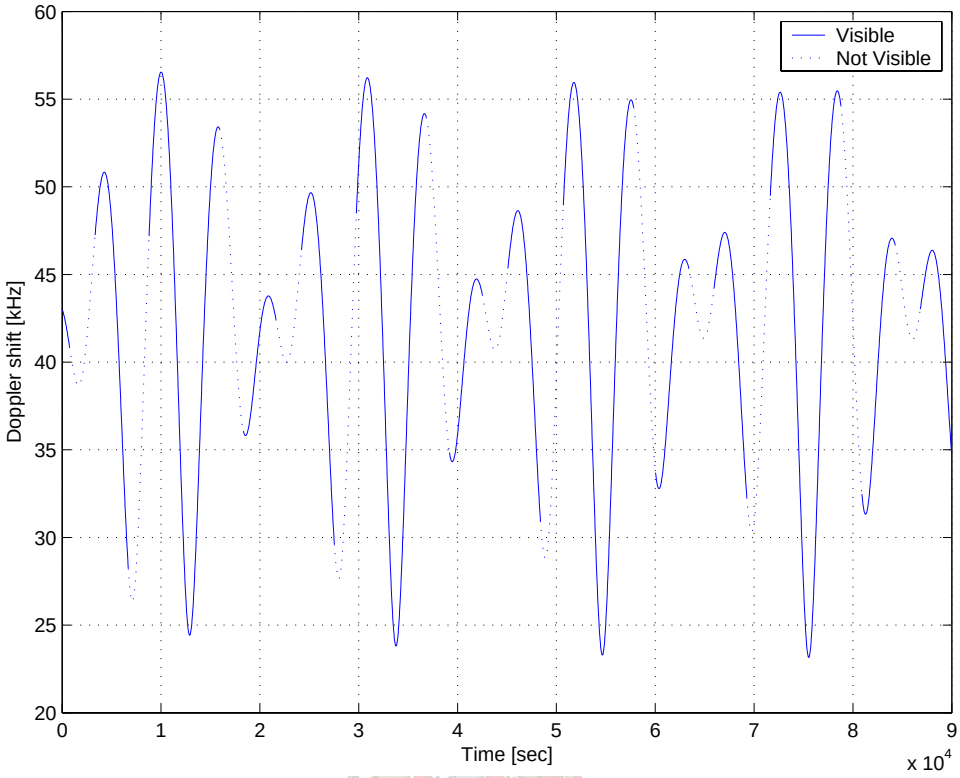


Figure 3.11: *TUBSAT simulation*

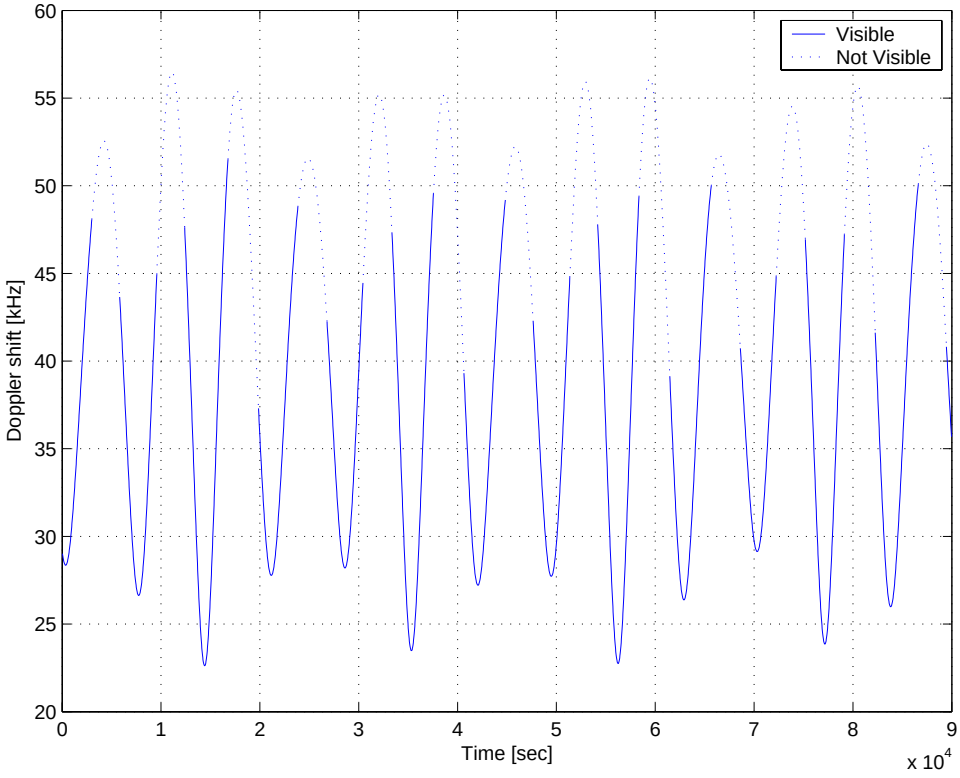


Figure 3.12: *SUNSAT simulation*

3.4.3 Geostationary Receiver Operation

Geostationary visibility

A geostationary satellite's altitude is 35 786 km, which places it in a higher orbit than the GPS satellites (altitude 20 000 km). The GPS satellite antennae are pointed toward the earth's centre and no signal is transmitted in the -Z (away from earth's centre) direction. Thus for a geostationary satellite to receive a signal, it has to utilise the part of the signal that leaks over the earth's edge. Thus the geostationary satellite has to lie opposite the GPS satellite and inside the GPS main beam limit (±21.3° [15, p. 80]) and the edge of the earth (13.87° off the satellite antenna boresight).

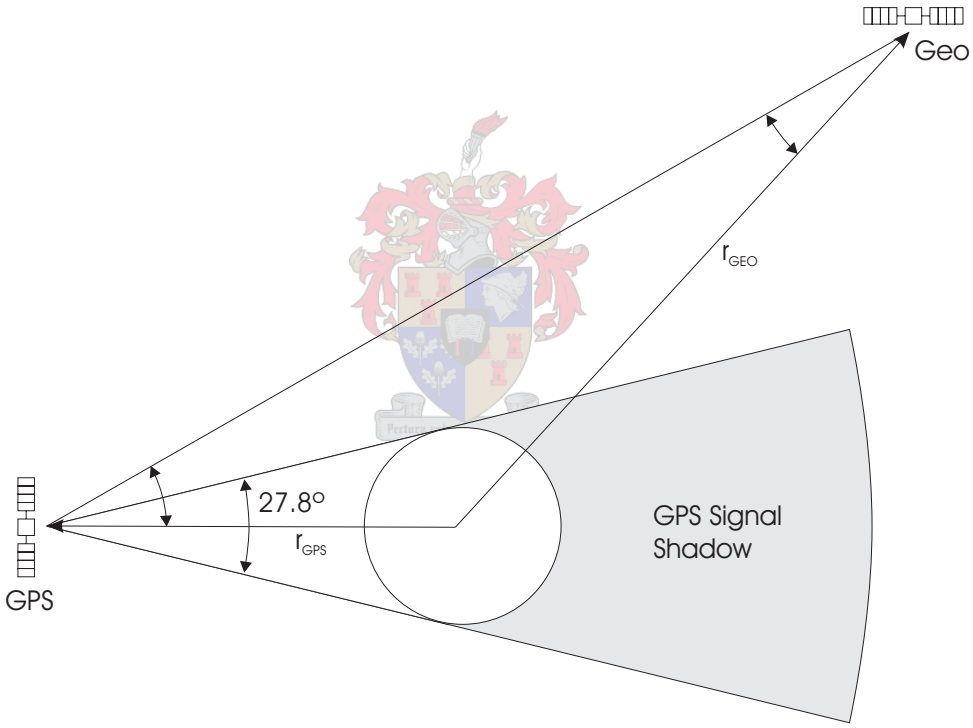


Figure 3.13: *GPS and Geostationary satellite angles (not to scale)*

Referring to Figure 3.13, to determine the angle between the GPS satellite boresight and the geostationary satellite, some elementary vector algebra is required. The angle θ between two vectors, $\mathbf{A}$ and $\mathbf{B}$ is given by:

$$\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{|\mathbf{A}|\,|\mathbf{B}|} \tag{3.13}$$

Which yields

$$\phi = \arccos\left[\frac{(-\mathbf{r}_{GPS})\cdot(-\mathbf{r}_{GPS}+\mathbf{r}_{GEO})}{|-\mathbf{r}_{GPS}|\,|-\mathbf{r}_{GPS}+\mathbf{r}_{GEO}|}\right] \tag{3.14}$$

from Figure 3.13 vector **A** is from the GPS SV position to the earth's centre ($-\mathbf{r}_{GPS}$) and **B** from the GPS SV to the geostationary satellite ($-\mathbf{r}_{GPS} + \mathbf{r}_{GEO}$).

If the signal passes close to the earth or through the troposphere, there is a path delay which must be considered in the navigation fixes. The atmosphere also "bends" the signal, implying that the pseudorange value is not a straight-line value.

Simulation

Geostationary satellites can also utilise the GPS side lobe [16, Figure 5]. Using this figure as a typical GPS transmitter antenna gain model and 0dB gain levels, two simulations were done to determine GPS satellite visibility for a geostationary satellite. The first using only the main lobe and a -3dB cut-off beam angle of 22° and a minimum angle of 13.9° (the earth's edge); the second including the side lobe at angles between 30° and 36°.

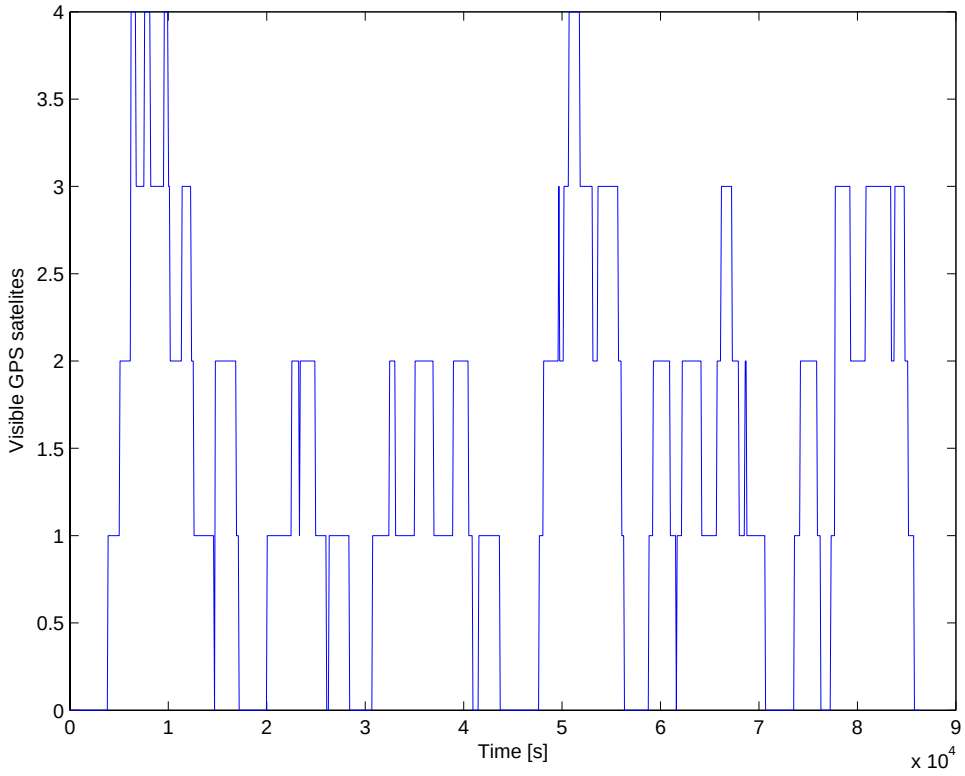


Figure 3.14: *Main Lobe simulation*

Figures 3.14 and 3.15 show the results of the simulations. The calculations were for a period of 24 hours at 86.4 second intervals. If only the main lobe is considered, enough GPS satellites are not visible for a position fix. However, if the side lobe is used, at least 4 satellites are visible for 94.4% of the time which will yield a navigation fix.

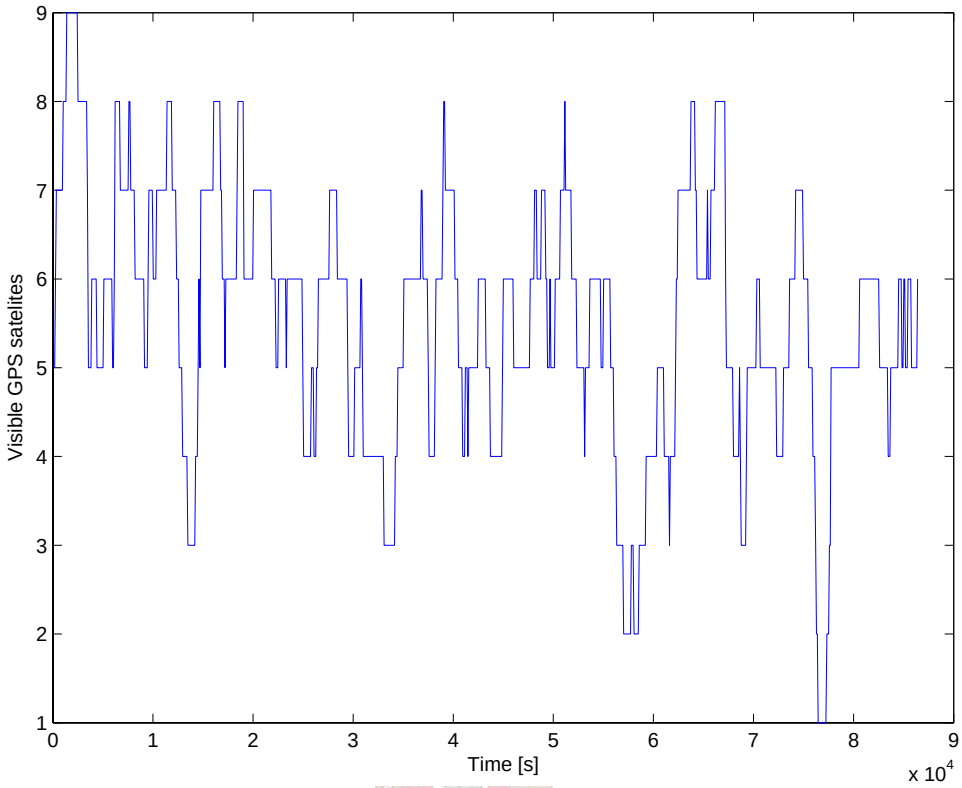


Figure 3.15: *Main and Side Lobe simulation*

Signal strength may be a problem because of the long distances, if the attenuation is higher than that of the earth's atmosphere.

3.5 Conclusions

3.5.1 Typical dynamic parameters

The simulations and calculations show that a receiver in a typical LEO environment has to contend with large Doppler shifts compared to those for terrestrial operation. These frequency shifts are in the order of ± 60 kHz for the L1 carrier frequency. By using orbital prediction models like SGP4 the Doppler shift that the receiver should expect from each SV signal can be accurately estimated. This estimate is used by the acquisition and tracking software.

3.5.2 Orbit estimation accuracy and element update frequency

From [11] the effect of TLE degradation on the Doppler frequency prediction can be limited to within ± 150 kHz if the elements are updated once per week, and new TLE elements can be easily uploaded as part of the satellite's general housekeeping programme. From previous work [11], and results obtained during testing (see Chapter 8), the SGP4 model is accurate enough to aid GPS signal tracking for warm or hot starts.

3.5.3 Terrestrial vs. Spaceborne Receiver Operation

At first it seems that the receiver has to operate in a harsh dynamic environment, but there exist many advantages in operating a receiver in space and more specifically on a satellite. These include:

- A satellite's position can be estimated accurately with an orbital model and in turn these estimates can be used in the receiver software to facilitate fast acquisition. In a terrestrial receiver the whole Doppler range is searched for the GPS signals until lock is achieved during a "Cold Start". (An estimate of the user position and velocity can be used by considering the last used values for a "Warm-" or "Hot Start", see Section 2.5). This may take a long time but the receiver only needs to search in a ±5 kHz range compared to a ±60 kHz range for a LEO receiver. But in a terrestrial receiver the user position and velocity and GPS almanac can be completely unknown and hence the long seek times. If the receiver is flown on a satellite, the position and velocity of the receiver can be accurately estimated — the entire Doppler range is not searched.

- The onboard clock of a satellite can be used to program the GPS. Then the receiver can predict its orbit and subsequently the expected GPS constellation state — a sky search for all SVs and all Doppler offsets is not necessary.

- The current GPS almanac can be uploaded to the satellite and then to the receiver. Terrestrial receivers are usually stand-alone devices.

- The along-track velocity and radial acceleration of the satellite is very accurately predictable, and the cross-track velocity and acceleration are very near zero.

- In space the atmospheric model can be significantly simplified from the complete atmospheric model but not completely ignored, although the effect of the atmosphere

on the signal is much less than for that of a terrestrial receiver.

- With the correct antenna and RF front-end design a receiver on a LEO satellite can receive signals from satellites with "negative" elevations.

Some disadvantages for a LEO receiver are:

- The obvious large signal dynamics are problematic for the acquisition and tracking software and adaptations are necessary.
- The receiver processor has an extra calculation burden to estimate the receiver's position and velocity, although the load on the processor is quite small [11] and should not impair receiver operation.

In the next chapter the GPS signal structure and receiver architecture will be discussed.

Chapter 4

GPS Receiver Architecture

4.1 Introduction

Currently the GPS system delivers 2 signals, the L1 and L2 signals, which are on different frequencies. Additional frequencies are being added in the near future which will yield more precise civilian position determination and added military functionality.

This chapter includes:

- An examination of the GPS L1 signal structure as the focus of this study is on single-frequency receivers. (Section 4.2).
- A discussion on how the code and carrier tracking loops operate. These function to remove the code and carrier from the signal and maintain lock on the signal from a particular SV (Section 4.3).
- A GPS receiver model. A model of the loops was built, first using MATLAB's Simulink and then in C++. A sample of the received Radio Frequency (RF) signal is generated and can be examined for correlation code sequences. With this model different code and carrier tracking loops can be experimented with. This could be useful when examining how high signal dynamics like Doppler shift rate affects the receiver and whether tracking can still be achieved in extreme dynamic environments. (Section 4.4).

4.2 GPS Signal Structure

4.2.1 L1 Signal

The L1 frequency is modulated with two separate codes with different chipping rates [1], the coarse/acquisition (C/A-code) code and the precise code (P-code) [7]. The P-code can be changed to the Y-code by the ground segment which is an encrypted code and for military and authorised user's use only. More precise calculations can be made with the P(Y)-code because of the higher chipping rate. Low-cost receivers utilise only the L1 signal and it will be the focus of discussion.

The C/A code's chipping rate is 1.023 MHz and the P(Y) code's rate is 10.23 MHz. The 50 bps data is combined with the code during modulation with an exclusive-or process denoted by a $\oplus$. The [P(Y)-code$\oplus$data] is modulated 90 degrees out of phase with the [C/A-code$\oplus$data] on L1, giving a quadriphase-shift-key (QPSK) modulated signal. The signal structure is illustrated in Figure 4.1 and the L1 signal is constructed as follows [7]:

$$L_{1i}(\omega_1 t) = A\left[P_i(t) \oplus D_i(t)\right]\cos(\omega_1 t) + \sqrt{2}A\left[G_i(t) \oplus D_i(t)\right]\sin(\omega_1 t) \tag{4.1}$$

Where the subscript i denotes the particular SV. Note that the C/A code amplitude is $\sqrt{2}$ the amplitude of the P(Y) code.

4.2.2 PRN codes

The C/A is a Gold code with a 1023 bit sequence length described in [4] and [7]. The chipping rate is 1.023 MHz which gives a code period of 1.0 ms. Each Gold code is derived from two code generators and the outputs are delayed and combined via an exclusive-or operation. Each generator (G1 and G2) is a 10-bit shift register which generates a pseudonoise (PN) code with a length of $2^{10} - 1 = 1023$ bits (the all zero state is an illegal state because the shift register stays in that state indefinitely, hence the -1). The polynomials for the two shift registers are: $G1 = 1 + X^3 + X^{10}$ and $G2 = 1 + X^2 + X^3 + X^6 + X^8 + X^9 + X^{10}$ [15].

The Gold code is obtained by exclusive-or'ing the G1 output with a delayed version of G2. The delay effect is obtained by tapping and exclusive-or'ing two bits in the shift

[1] The term "chipping rate" is used instead of "bitrate" because the code contains no information

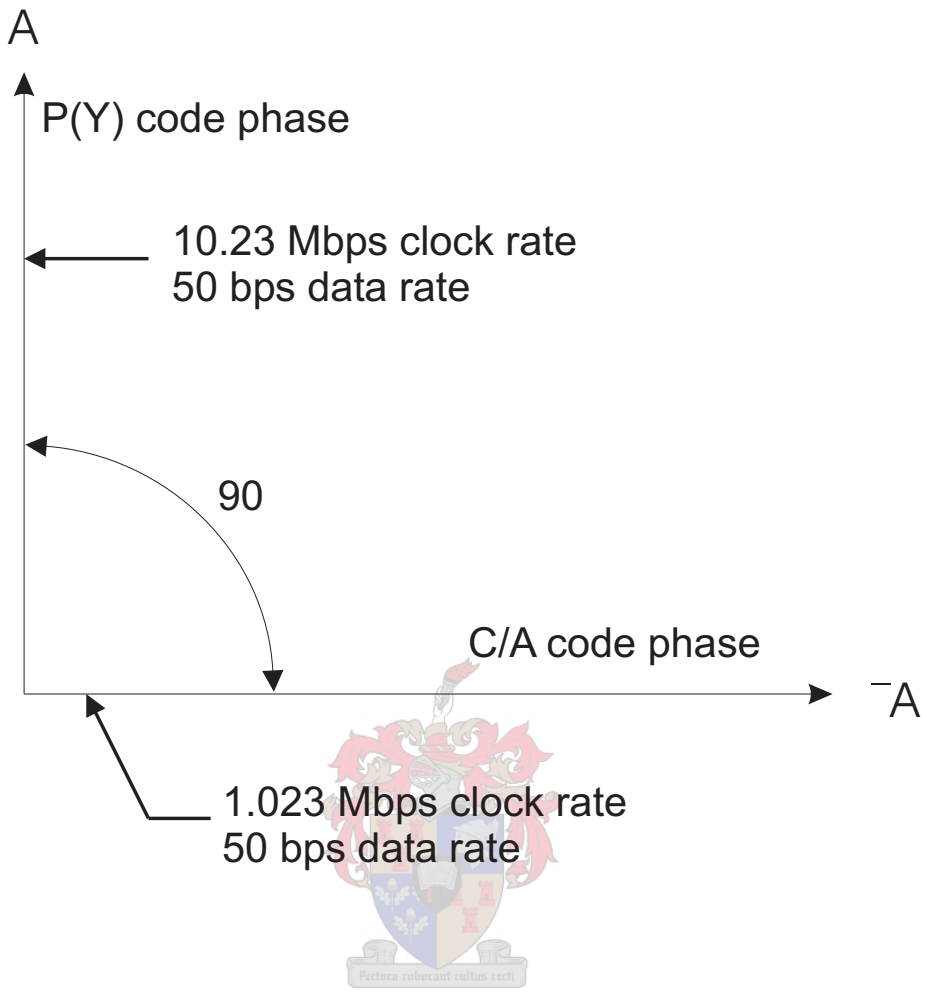


Figure 4.1: *L1 signal structure*

register G2. This is because the PN code sequence has the property of not changing when a phase-shifted version is added to itself, it simply obtains another phase.

The tap selections, subsequent delays and first 10 chips of the PRNs 1 to 5 are listed in Table 4.1. The full table is listed in Appendix B.1. Figure 4.2 shows how the codes are generated with shift registers.

Table 4.1: *Code-phase assignments for the C/A-Code*

SV PRN Number	C/A-Code Tap (Chips)	C/A-Code Delay (Chips)	First 10 C/A-Chips (Octal)*
1	$2 \oplus 6$	5	1140
2	$3 \oplus 7$	6	1620
3	$4 \oplus 8$	7	1710
4	$5 \oplus 9$	8	1744
5	$1 \oplus 9$	17	1133
.	.	.	.
.	.	.	.

The MATLAB code to generate the C/A codes can be found in Appendix B.2

4.3 Code and Carrier Tracking

The GPS signal is modulated by both the carrier and the PRN code. To reconstruct the data message the carrier and code signals have to be stripped from the signal. Both the carrier and code tracking loops have to maintain a lock on their respective signals. Figure 4.3 [7] is a illustration of how the sampled digital IF is demodulated and input into the receiver processor, which extracts the navigation message and takes the pseudorange measurements. E denotes a code which is 1/2 chip early, P is in code phase and L is 1/2 chip late.

The signal search across the Doppler shift range is performed by the receiver processor - it guides the centre frequencies of the numerically controlled oscillators (NCOs) and code offsets until lock is achieved. Figure 4.4 shows the presence of a signal at a certain code offset and Doppler frequency offset — a 1 msec sample signal was correlated with one period of the PRN code with carrier removal at different frequency offsets. It is clear that only at a certain carrier frequency and code phase will a correlator indicate that a signal is present.

*In the octal notation for the first 10 chips of the C/A code as shown in this column the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. For example, the first 10 chips of the SV PRN number 1 C/A-code are 1100100000.

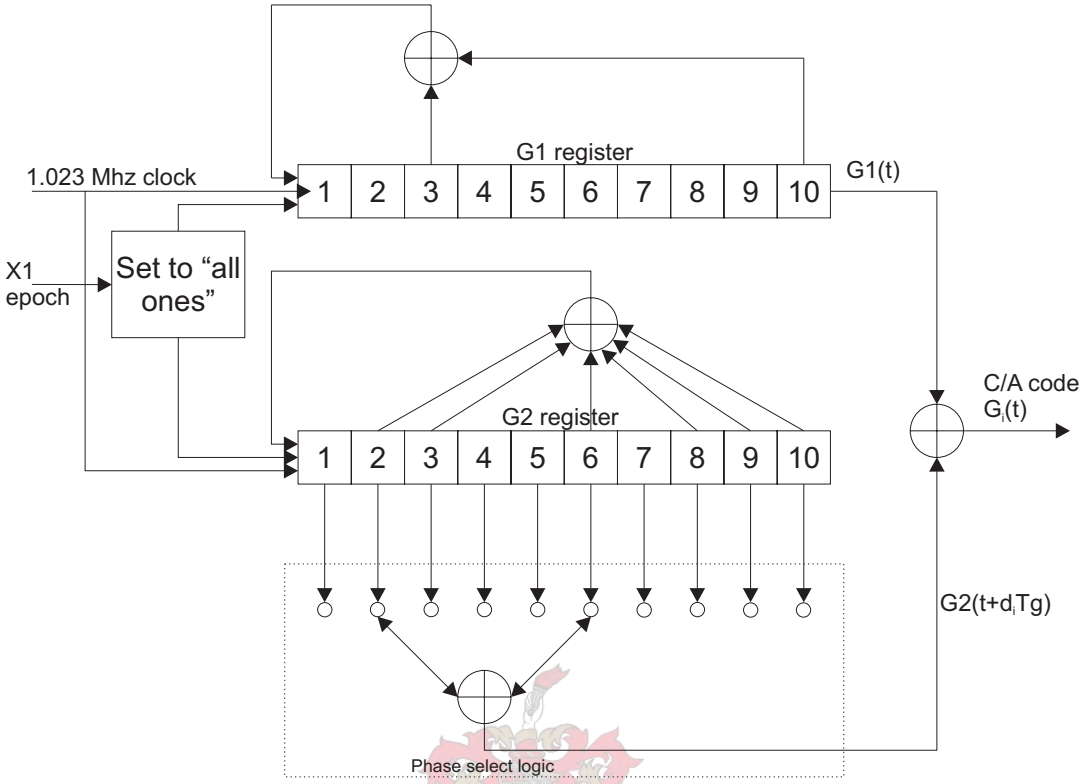


Figure 4.2: *PRN C/A-code generator*

After the carrier and code signals have been removed, the In-phase (I) and Quadrature (Q) components are formed. These are used in the discriminators to maintain lock and is used to calculate the signal amplitude, $\sqrt{I^2 + Q^2}$ [8]. The amplitude is compared with a threshold to determine whether lock has been achieved. Figure 4.5 [7] is a block diagram of how carrier and code tracking is performed by the receiver processor.

Section 4.2.1 showed that the signal has both an in-phase and quadrature component. From eq. 4.1 the C/A signal is modulated with $\sin(\omega_1 t)$ and the P(Y) component with $\cos(\omega_1 t)$ signal to construct the L1 signal. Thus to recover the C/A$\oplus$code signal the digital IF is mixed with the sin signal to obtain the in-phase (I) component. The quadrature (Q) component contains the P(Y) signal.

In a single-frequency receiver the digital IF sampling frequency is usually lower than the Nyquist frequency required to recover the P(Y) signal. For example, in the GP4020 chipset, the sampling frequency is 5.714 MHz and the Nyquist frequency required to sample the P(Y) code is 20.46 MHz — the P(Y) signal cannot be recovered from the quadrature component and the Q-component contains only noise [8].

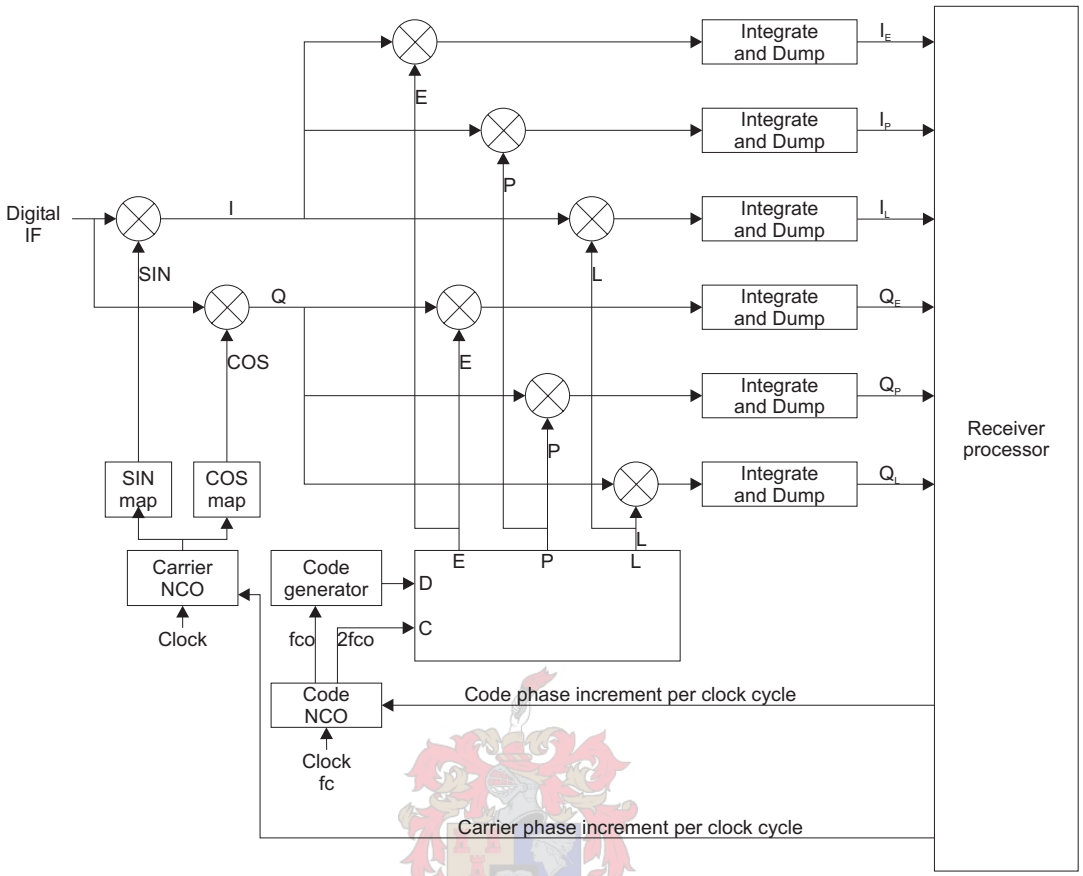


Figure 4.3: *Receiver Block Diagram*

4.4 GPS Receiver Model

4.4.1 Introduction

The carrier and code tracking loops play an important part in signal acquisition and maintaining lock on the signals. Together with the receiver processor the received digital IF has to be searched for valid signals and lock has to be maintained on the separate SV signals. Most receivers contain quite a few identical correlator channels — some can be locked onto a signal while others are searching for signals depending on the current state of the receiver.

The software contained in the receiver processor is responsible for assigning the channels to their individual functions. The channels are programmed with the frequencies to be searched as well as the code offsets. One correlator channel was implemented first in MATLAB's Simulink, and thereafter in C++. These models can be used to evaluate different types of tracking loops with a simulated signal or with a signal sampled from a

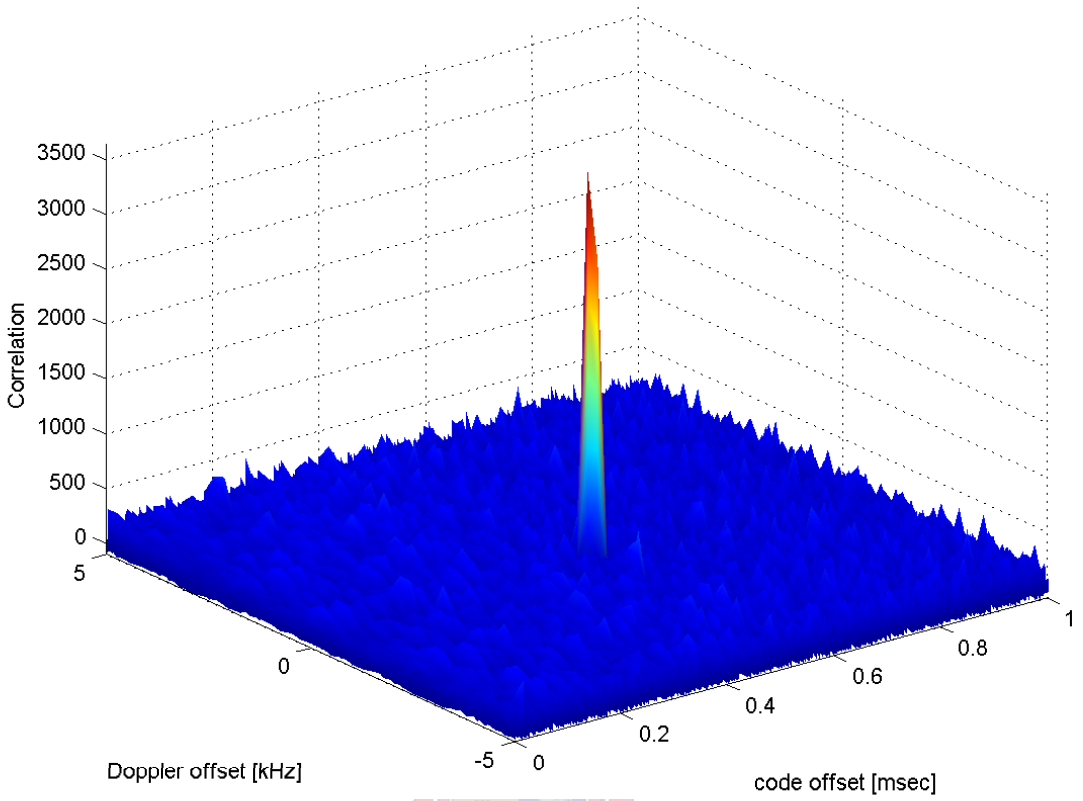


Figure 4.4: *Sample signal correlation results*

GPS receiver.

4.4.2 Signal Generation

MATLAB was used to simulate the digital intermediate frequency (IF) signal. A sample time is chosen and the signals constructed as they would in a real GPS receiver. The amplitudes of each signal can be adjusted to simulate free space losses. For testing purposes all the GPS signals were of the same amplitude and random noise added.

Each SV signal is modulated with a random code and carrier phase, as the arriving signals will have an unknown code and carrier phase. Then the individual QPSK signals are added together to simulate the received IF. The MATLAB code for generating the signal is listed in Appendix G.1.

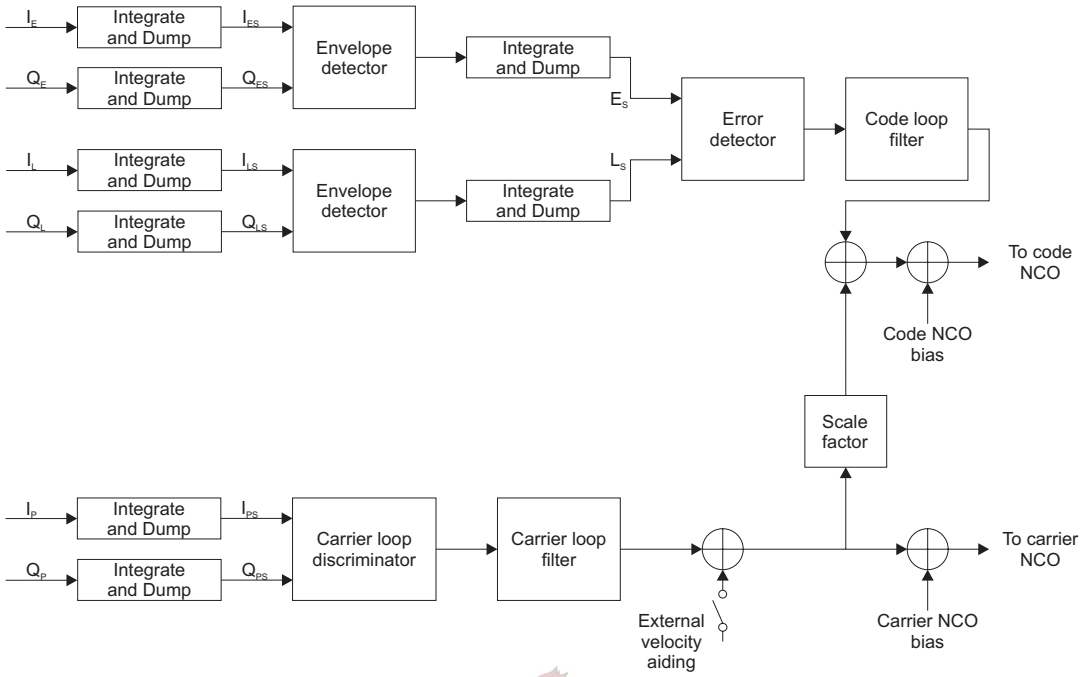


Figure 4.5: *Tracking loop diagram*

4.4.3 MATLAB Simulink Model

Figure 4.6 is the correlator channel implemented in Simulink. The received signal is input into the carrier and code tracking loops. The code tracking loop is set to initially contain the correct code phase of the wanted signal. Usually the digital IF has a two-bit quantisation which improves the channel's ability to lock onto the signal. The quantisation can be set when generating the simulated signal.

The Simulink subsystems (`CodeGenerator, Correlate&Hold, Sin&CosVCO` and `Filter&Integrate`) are shown in Appendix G.2. The code generator block contains an Numerically Controlled Oscillator (NCO) which responds to the code tracking loop discriminator to steer the code phase. The discriminator algorithm implemented for the code loop is the "Dot product power discriminator" [7]. It is the only discriminator that uses all three correlators and results in the lowest computational load. Other discriminators can be easily implemented by editing the `Correlate&Hold` block.

The carrier is tracked by using a Costas phase lock loop (PLL) with a product discriminator [7]. The Costas PLL is used because it is insensitive to 180 degree phase changes in the I and Q signals. This is necessary since the navigation message is the output on the I channel and the phase will change with data bit transitions.

The result of a sample signal can be seen in Figure 4.7.

4.4.4 C++ Implementation

The Simulink model is quite handy for examining the correlator channel operation. However, it can become quite cumbersome and slow; even on quite a powerful PC. The model can be implemented in C++ for the following reasons:

1. Speed: The Simulink model is very slow. It can take minutes to simulate a signal of only a few milliseconds. The C++ implementation would be much faster.
2. Configurability: The C++ model can be configured quite easily. It can be used with different input files for different filters etc., without the need to edit each individual block. The C++ model's parameters, filters and discriminator algorithms can be altered **during simulation**.

Each block of the model was implemented from scratch. To test the implementation the simulation was run with the same sample signal input as the Simulink model. Figure 4.8 shows the results.

4.4.5 Conclusion and Recommendations

The channel model can be used for evaluating tracking loops with simulated or real digitised signals. It is however too slow at this moment to be implemented as a real-time receiver. The Simulink and C++ model show the same results for the same input signal.

Several aspects of the model can be examined and optimised:

- The integration times at the moment are quite arbitrary and do not correspond to real-world values. The filters and loop gains have been determined empirically — these should be calculated theoretically.
- The noise added to the signal is not bandwidth limited, just random white noise. In the receiver the noise will be bandwidth limited as the signal passes through bandpass filters.
- Higher quantisation levels will improve the quality of operation and is application specific. Commercial receivers use 2-bit quantisation

In the next chapter the hardware choices and a proposed multiple-receiver design will be discussed.

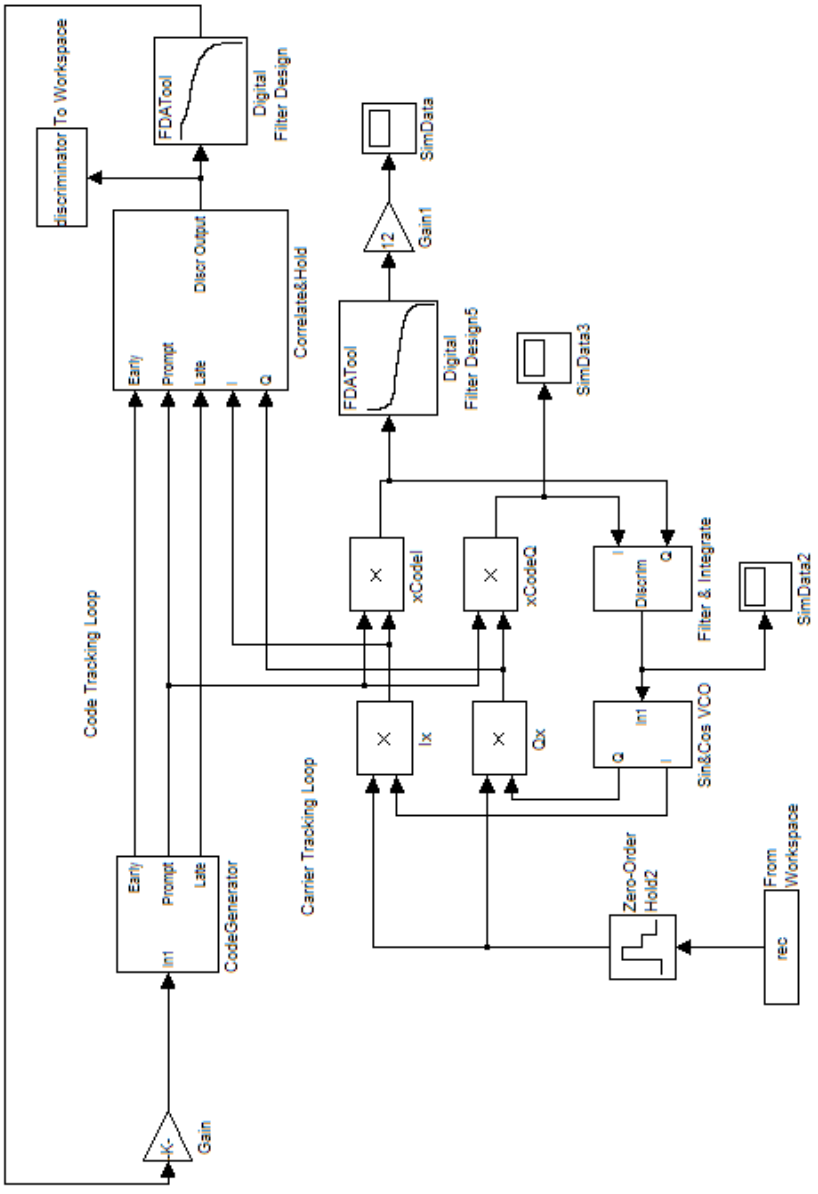


Figure 4.6: *MATLAB Simulink Correlator Channel Model*

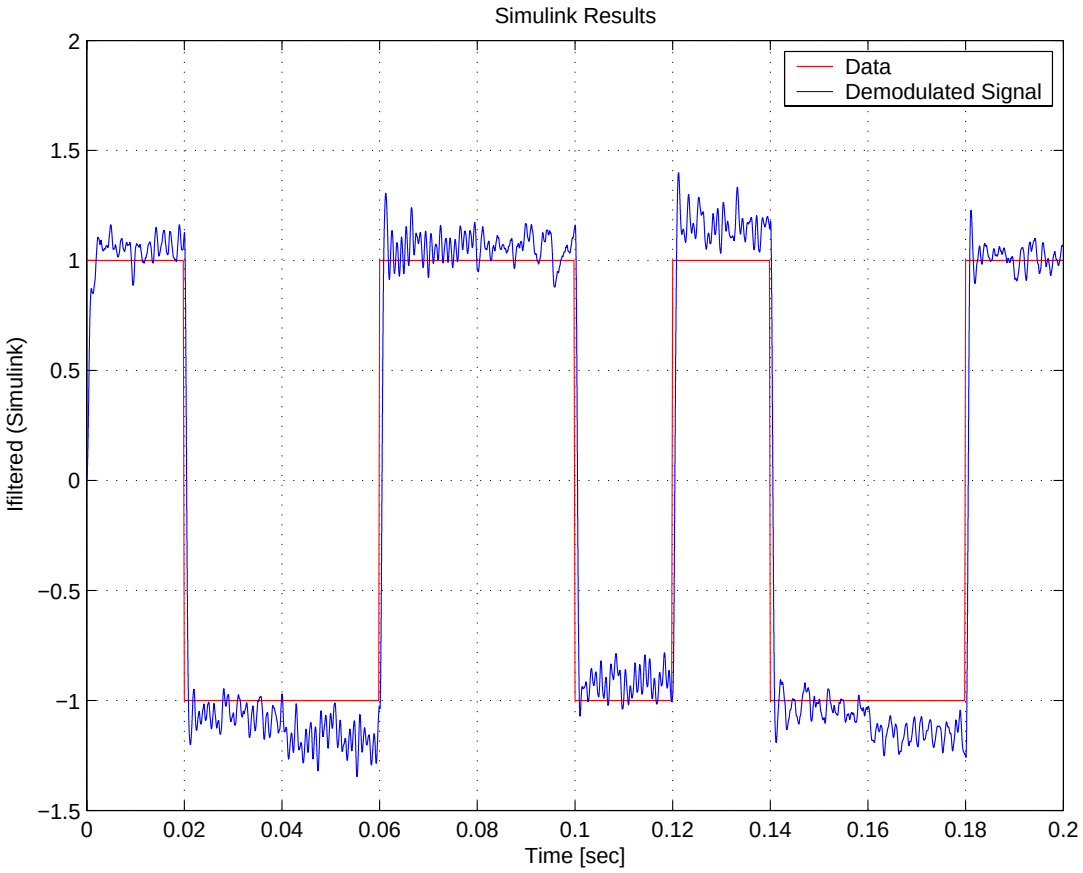


Figure 4.7: *Simulink Model Output*

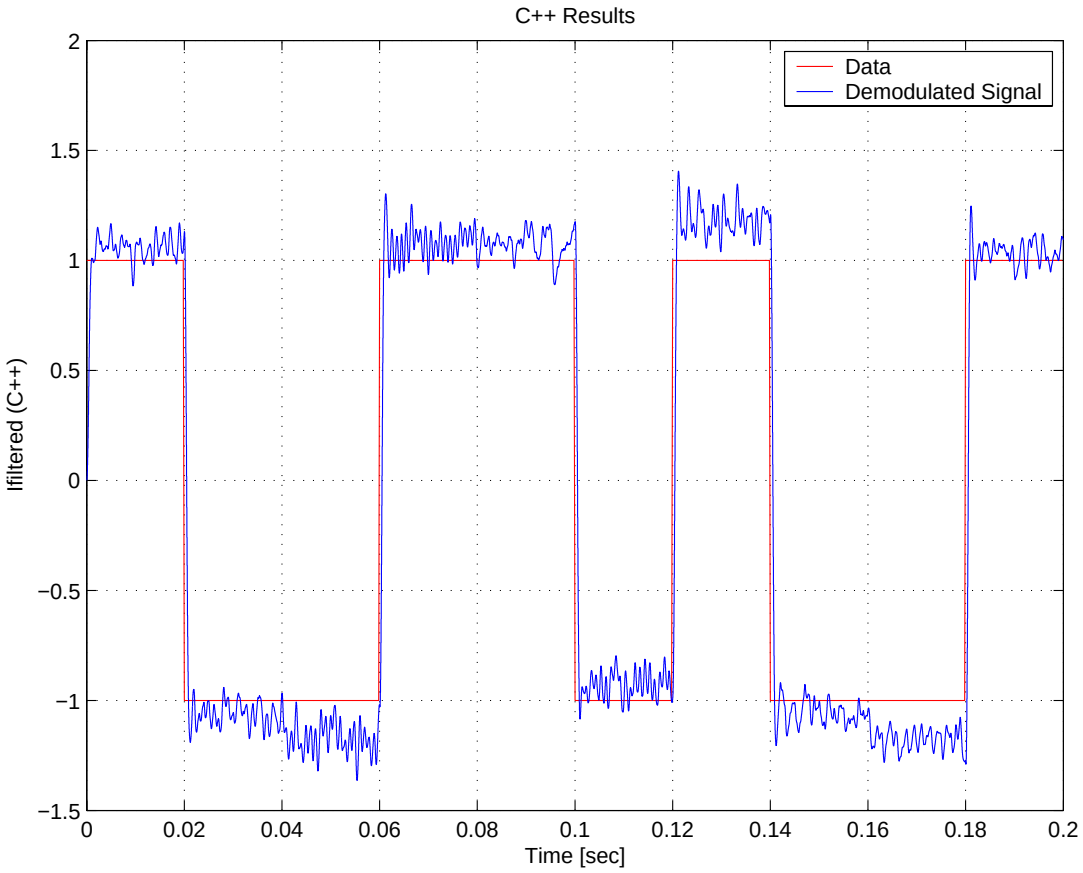


Figure 4.8: *C++ Model Output*

Chapter 5

GPS Hardware

5.1 Introduction

In this chapter the hardware issues of the SGPS receiver will be discussed. They are:

- The factors driving the hardware selection process and the hardware that was acquired (Section 5.2).
- A proposed 4-antenna design which can be used for attitude determination (Section 5.3).

5.2 Hardware Selection

5.2.1 Development kits and existing spaceborne receivers

The hardware selection was driven by two factors:

- Obtaining a development kit that would enable us to modify the GPS software so the receiver could operate in space, i.e. the source code has to be supplied with the development kit
- Using a chipset that has been proven to be space-capable.

There are not many GPS development kits that enable the user to modify the source code and add functionality to the software, and not many spaceborne receivers exist. The

ORION GPS receiver has been proven in space ([14] and [11] and [21]) and is based on the Mitel (now Zarlink) GP2000 chipset. The GP2000 chipset comprises the GP2010 or GP2015 RF downconverter, the Dynex Semiconductor DW9255 35.42MHz (2MHz BW) IF SAW filter, the GP2021 correlator IC, and the 32-bit ARM60-B RISC processor. The ORION receiver implements the GP2000 chipset [29].

The chipset has been replaced with the GP2015 RF front-end [26] in combination with the GP4020 GPS Receiver Baseband Processor [28]. The GP4020 combines the 12-channel correlator function of the GP2021 with an advanced ARM7TDMI (Thumb) microprocessor to achieve a higher level of integration, reduced system cost, reduced power consumption and added functionality. Unfortunately the newer chipset does not offer input from two antennas as the GP2021 correlator had, and the integration of the correlator and processor make it more difficult to design a 4-antenna receiver. When designing a multiple-antenna receiver for attitude determination purposes, four baseband processors have to be used (see Section 5.3).

Development kits for the GP4020 chip are available from Sigtec Navigation (SIGNAV) and include most of the source code for the receiver software. The GPS Architect source code is available in full source form for the GP2000 chipset. It was decided to obtain a development kit from Sigtec Navigation, together with the GPS Architect source code from Zarlink Semiconductor. The software has to be ported to the newer chipset (see Chapter 6) and the Sigtec Navigation MG5001 OEM Board is the hardware proposed to be used in the GPS subsystem.

5.2.2 Power Consumption and Receiver Volume

Power consumption and hardware volume are important considerations on a satellite. The SigNav MG5001 uses considerably less power than the ORION design, and is much smaller. The MG5003 (also available from Sigtec Navigation) is even smaller that the MG5001, as it is designed for cellular phone applications. From [29] and [19] the power consumption and physical dimensions are listed in Table 5.1.

The newer chipset offers a smaller volume, lower power solution. The MG5003 offers only one serial port (no RTCM correction data), but in space there are no differential corrections available, so the MG5003 could be used as a SGPS receiver.

Table 5.1: *ORION and MG5000 Power consumption and physical dimensions*

	ORION	MG5001	MG5003
Power Consumption			
Min Voltage [V]	5	3.3	3.3
Current [mA]	395	110	110
Power [W]	1.98	0.363	0.363
Physical Dimensions			
Length [mm]	95	69.85	38.1
Width [mm]	50	46.36	27.94
Height [mm]	20	11.07	11.07
Volume [cm^3]	95	35.8	11.8

5.3 Multiple-antenna design

5.3.1 Motivation

The main use of the GPS system is for determining position and velocity of a receiver. However, it can also be used for attitude determination, a very important aspect of satellite control. This is achieved by using the carrier phase measurement of the received signal. If the distance between two antennae is known, the carrier phase difference can be used to determine the angle between the vector connecting the two antennae, and the vector to the GPS satellite.

For example, if the carrier phase of both antennae are the same (and the distance between the two is less than the wavelength to avoid ambiguity), the antenna vector is perpendicular to the GPS satellite position vector. If the antennae are a half wavelength apart and the carrier phase difference is 180°, the antenna vector is parallel to the GPS satellite position vector.

If four antennae with known positions on a object (e.g. a satellite) are used, these angles can be used to determine the attitude of the object in the ECEF reference frame. This section proposes a 4-antenna design based on the Zarlink chipset. Four receivers are required, since 3 only yield an unambiguous 2-dimensional plane and the solution is the angle between the signal vector and the plane normal — rotation about the normal is unobservable.

5.3.2 Hardware Requirements

Chipset

The GP2000 chipset is designed for a single-antenna solution. The newer chipset based on the GP4020 (see Section 5.2 page 54) is also designed for a single-antenna receiver. If a 4-antenna receiver is to be implemented with the GP4020, four separate receivers have to be used. A single-processor design would be much simpler to implement.

A single processor design can be implemented on GP2000-based design. The single-antenna solution consists of a GP2015 RF downconverter, GP2021 correlator IC and the 32-bit ARM60-B RISC processor. Figure 5.1 depicts the architecture.

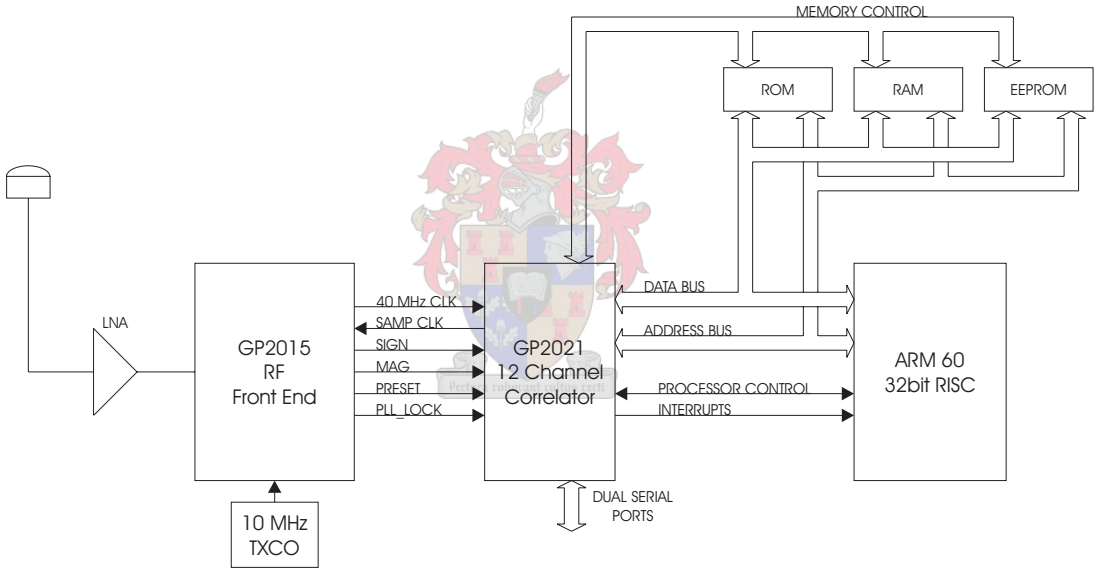


Figure 5.1: *GP2000 Single Antenna Design*

The GP2021 correlator IC can accept input from 2 RF front-ends via the SIG0/MAG0 and SIG1/MAG1 pair. The source for each tracking channel is determined by the `CHx_SATCNTL` register [27] for each channel. By combining 4 RF Front-ends with 2 Correlator ICs, 24 tracking channels are obtained from 4 antennas. These channels can be assigned as determined by the processor, keeping in mind that as more channels are assigned to one antenna, the other antenna in the pair is assigned to less channels. A maximum of 12 channels can be assigned to one antenna, with the other antenna in the pair feeding no tracking channels.

A single processor can be used to control the whole system. This makes the system much simpler regarding channel assignments and attitude determination.

Integration

The proposed 4-antenna design is shown in Figure 5.2.

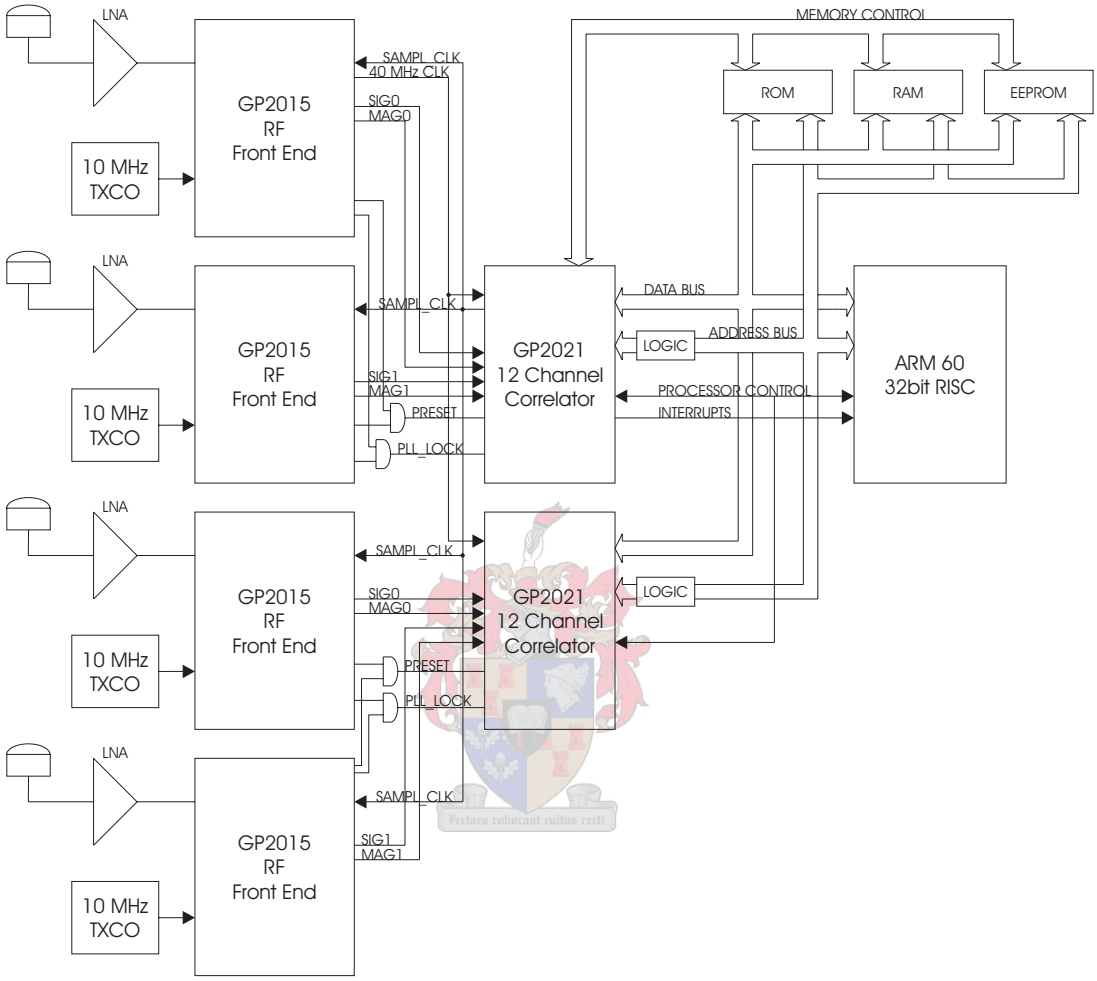


Figure 5.2: *GP2000 4-Antenna Design*

A single Temperature Compensated Crystal Oscillator (TCXO) can be used for the RF Front-ends for down-conversion of the signal to a 4.309MHz IF [26]. The same 40 MHz clock is used in both correlators, making sure that buffering is used as required. The sample clock (SAMPL_CLK) is derived from the primary correlator and input to all 4 RF Front-ends, again noting that buffering has to be used if required.

Both correlators have memory-management functions, but only one memory manager is required. From [27] A[22..20] is used to select the external memory devices or internal correlator registers as listed in Table 5.2.

Table 5.2: *ARM system address map*

A22	A21	A20	Device selected	Decoded output pin
0	0	0	ROM	NROM
0	0	1	RAM	NRAM
0	1	0	Correlator	
0	1	1	Support Functions	
1	0	0	EEPROM	NEEROM
1	0	1	User Defined	NSPARE_CS
1	1	0	Not Decoded	
1	1	1	Not Decoded	

Addressing

The ARM processor has a 32-bit address bus, and [25] shows that only 23 bits of the address bus is normally used. Bits [22..20] are used by the correlator to select the memory device as mentioned. Bits [2..18] are used to address the external memory (2^{17} = 128K 4-byte words = 512Kb). The extra address bits can be used to select between the correlator ICs. A 0 on bit 23 indicates that the primary correlator is addressed, accessing the internal registers or the memory manager functions. A 1 on bit 23 indicates that the registers of the secondary correlator are addressed. The memory management functionality on the secondary correlator is not used, all memory access are done through the primary correlator.

Some logic is required to implement the addressing. When bit 23 on the address bus is high, A[22..20] on the primary correlator are set to all ones, since this is not decoded (see Table 5.2). A[22..20] on the secondary correlator are set to either [010] or [011], depending on whether the internal register or support functions are addressed. When A23 is low, A[22..20] on the primary correlator is assigned normally, and A[22..20] on the secondary correlator is now set to all ones. Table 5.3 shows the truth table.

Each RF front-end has a PReset and PLL output. The Preset output is low while the power is switching on, and goes high when a nominal power supply value has been achieved [26]. The two Preset outputs are combined by an AND gate, i.e. the correlator will switch on when both the front-ends are powered up. The same is done for the PLL LOCK outputs.

Table 5.3: *Addressing truth table*

Address Bus				Primary Correlator			Secondary Correlator		
A23	A22	A21	A20	A22	A21	A20	A22	A21	A20
0	**0**	**0**	**0**	**0**	**0**	**0**	1	1	1
0	**0**	**0**	**1**	**0**	**0**	**1**	1	1	1
0	**0**	**1**	**0**	**0**	**1**	**0**	1	1	1
0	**0**	**1**	**1**	**0**	**1**	**1**	1	1	1
0	**1**	**0**	**0**	**1**	**0**	**0**	1	1	1
0	**1**	**0**	**1**	**1**	**0**	**1**	1	1	1
0	**1**	**1**	**0**	**1**	**1**	**0**	1	1	1
0	**1**	**1**	**1**	**1**	**1**	**1**	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1
1	**0**	**1**	**0**	1	1	1	**0**	**1**	**0**
1	**0**	**1**	**1**	1	1	1	**0**	**1**	**1**
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

5.3.3 Software Design

Satellite Selection

A single-processor design is necessary to give sufficient control over the tracking channels. For attitude determination, at least 16 channels that are tracking the same four satellites are required. In this design there is 24 channels. For an optimal attitude solution, all 24 channels could be used to track six GPS satellites for an overdetermined solution, but no extra channels will be available to acquire other SV signals, and lock will be lost when a GPS satellite disappears over the horizon.

A proposed solution would be to use 20 out of the 24 channels to track 5 GPS satellites with each of the 4 antennae, and using the remaining 4 channels to maintain lock on 4 other GPS satellites. When a satellite signal becomes weak, the 4 channels assigned to it will switch over to a new satellite which gives the lowest position dilution of precision

(PDOP) (See Section 2.6). During the switch over period, 4 strong signals will still be tracked, but the attitude solution will be less accurate (4 instead of 5 satellites used in the solution).

Processing Power

The satellite selection and attitude determination algorithms will place an extra burden on the processor. The SGP4 orbit propagator already uses extra processing power. Because RTCM 104 differential correction (RTCM) is not used, this task can be omitted which frees up some processing cycles.

5.4 Conclusions

There are many implementations of GPS receivers, ranging from single-frequency, single antenna devices to dual frequency, multiple antenna designs. In the near future there may soon exist dual GPS/Galileo receivers (Galileo is the intended European Satellite Navigation system). A receiver should thus be chosen by the accuracy or extra functionality required.

In the next chapter the software port to the SIGNAV hardware will be discussed.

Chapter 6

GPS Architect Software Port

6.1 Introduction

As described in Section 5, the GP4020 chipset was selected as a hardware platform for the SGPS receiver. The full source code is not available with the development kit, so the full GPS Architect source was acquired. The GPS Architect has been successfully modified to operate in space in the past [14] on the ORION receiver. Thus the decision was made to port the GPS Architect software to the GP4020 chipset.

There are two main issues when porting the software from the GP2021/ARM6 hardware to the GP4020 chipset.

- The GPS Architect software uses a simple time-slice operating system. The Real-time Operating System (RTOS) that was part of the SIGNAV receiver's software was kept. The reasons for this are given in Section 6.4. An overview of these two OSs are given in sections 6.2 and 6.3.
- The GP4020 chipset differs in hardware implementation from the GP2021/ARM6 combination and the software has to be adapted to these changes. Section 6.5 describes these changes.

6.2 GPS Architect OS

The GPS Architect employs a task-switching OS and supports a set of user-defined prioritised tasks. A task at any one time can either be Active (suspension interval is zero)

or suspended (suspension interval is non-zero), where the suspension interval is how long the task must wait before it is executed again. The primary control of the software is derived from the ACCUM_INT interrupt of the GP2021 which has programmed period of 900.025 μsec in the Architect software. This interrupt is used to update the tracking loops and to check for the occurrence of a TIC. The `accum_status_b` register is examined for a TIC occurrence — this signals the availability of new measurement data, and has a nominal period of 0.099 999 9 seconds.

[bufacc.c]

```
accum_status_b = (unsigned)inpf(ACCUM_STATUS_B);
if(accum_status_b&0x2000)
  {
    TIC++;
    EXECUTIC++;
  }
```

The `EXECUTIC` variable is used for task switching. The GP2021 also provides a MEAS_INT interrupt, which occurs together with the TIC, but in the Architect software this interrupt is not used.

On the occurrence of a TIC or when a task is suspended, any task which has a zero suspension interval is activated, according to priority. An active task which currently has control of the microprocessor will temporarily lose control if a higher priority task becomes active. This system implies that in the Architect software a task can only be delayed by integer amounts of the TIC period. When a task is suspended, the suspension interval is decremented on each TIC until it reaches zero and reactivates. If no other task requires processing time, the lowest-priority task remains active and cannot be suspended.

Each task has a entry in the Task Control Block (TCB) that specifies the entry function, stack size etc. The order in which the tasks are listed in the TCB determines the priority of execution. The `PROTECT` variable enables or disables task switching to prevent inconsistencies in shared variables and is a way to define critical sections in the code. Task switching can also be disabled by disabling the ACCUM_INT interrupt on the GP2021 with the `disable()` and `enable()` functions:

[io.c]

```
void disable(void)
```

```
{
  outpw(SYSTEM_SET_UP,MASK_INTERRUPT);            /* disable interrupts. */
}
void enable(void)
{
  outpw(SYSTEM_SET_UP,UNMASK_INTERRUPT);           /* enable interrupts. */
}
```

with the masks defined as appropriate to the GP2021 correlator IC.

The default tasks of the Architect software are as follows:

Name	Stack Size (bytes)	Suspention Interval [TICs]
TTakeMeas	2000	1
TNav	5000	10
TDisplay	9000	10
TRTCM	2000	3
TProcSbf	2000	10
TAlloc	9000	1
MAIN	0	N/A

Notice that the main task is the last defined, and lowest in priority. The task functions are:

- **TTakeMeas:** Controls the carrier frequency bin search and collects the raw observation measurement data from the tracking channels.
- **TNav:** Processes the satellite measurement data to perform the navigation solution. Updates the position and velocity fix and the receiver clock model.
- **TDisplay:** Construct the output messages that are transmitted via RS232. Proprietary WINMON messages and standard NMEA messages are available
- **TRTCM:** Detects the presence of DGPS corrections signal and parses the values into variables
- **TProcSbf:** Processes the received navigation message, extracting the data and storing as appropriate. Initialisation of software receiver clock offset to GPS system time model and maintains the model offset to within 0.5 seconds.

- **TAlloc:** Allocates satellites to the tracking channels according to the operation mode (be it Cold Start or Highest Elevation mode) and available data. Checks for received commands to the receiver; updates the estimate of the receiver oscillator error that is used to determine frequency search range

6.3 microITRON RTOS

The SIGNAV source code runs on a Real-Time Operating System (RTOS) called ITRON developed at the Sakamura Lab, University of Tokyo, Japan. ITRON is a real-time, multitasking OS specification intended for use in industrial embedded systems.

There are many advantages to using an OS that was specifically developed for use in embedded systems. On a personal computer, the main purpose of an OS is to protect data from different users and file system management [17], whereas on an embedded system the requirement is concurrent processing of tasks and interrupt handling from external events. The μITRON RTOS is a pre-emptive multitasking OS.

The two most important aspects of the RTOS applicable to the design is the task scheduling method and the messages which are sent between tasks (see Section 6.4).

Tasks

A "task" refers to a unit of concurrent processing. Algorithms and functions inside the task will execute sequentially, but the tasks execute "concurrently". From a GPS receiver point of view this is an important point as there are several processes that occur concurrently, e.g. extracting the navigation message from the signal, computing position and velocity solutions and outputting the data. These tasks are dependent on each other for correct receiver operation, but can be executed separately and data can be exchanged via shared memory. The OS is responsible for assigning processor time to each task using time sharing.

The "dispatcher" is the mechanism by which the task being executed is switched ("dispatched") and this is done by either a system call or an interrupt. Thus, a task switch can either occur if:

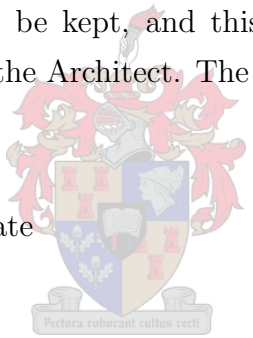
- A task has finished operation for the moment and a system call is executed. Usually this call includes the amount of time that the task should be dormant, or in a

wait state. The OS then decides according to priority which task should be given processor time.

- An interrupt occurs: The interrupt code is executed and again the task that should execute next is determined by priority. Interrupt handlers are considered task-independent portions under ITRON and μITRON specifications.

"Task Scheduling" is the manner in which the OS decides which task should be executed at which time. ITRON uses a priority-based scheduling, i.e. every task has a certain priority assigned to it, and the task with highest priority that is ready to execute will be run. A smaller priority value indicates a higher priority. The ITRON priority specification is absolute: as long as a task with a higher priority is executing, no task with lower priority will be allowed to execute.

Information for each task has to be kept, and this is done via a Task Control Block (TCB), similar to the method in the Architect. The TCB data structure contains all the following:



- Flags indicating the task state
- Priority
- Storage region for the program counter, general-purpose registers and task stack pointer
- Task start address

If more than one task has the same priority, the tasks are executed on a "First come, first served" basis. If task A and B have the same priority, and task B is pre-empted by an interrupt, task B will be given execution time again after the interrupt. If a higher priority task is executed, again, when the higher priority task is finished task B will be given processor time. Thus, when a task is pre-empted, it does not move to the back of the ready queue.

In the SIGNAV implementation, a timer interrupt is used for task pre-emption. More specifically, Timer B is implemented at a period of 20 ms. This differs from the Architect OS method of using the TIC (with a nominal period of 99.9999 ms) for task switching; more details can be found in Section 6.4.

The Timer 2 (TIC2) interrupt is enabled in the `InitIntC(void)` function in `intc.c` with the `SetupInterrupt()` function. In `cpuarm.s`, the `timer_handler` contained in `timer.c` is called when that interrupt occurs:

[cpuarm.s]

```
IrqTic1TOB
  BL        task_exit_count             ; Update task end time
  LDR       r0, =|timer_handler|
  MOV       lr, pc
  BX        r0
  B     ret_int                     ; 'ret_int' interrupt complete
```

The following excerpt from the μITRON specification is important: *Task switching (dispatching) is not performed while task-independent portions are executing. Even if the result of a system call issued is a dispatching request, that dispatching is delayed until control leaves the task-independent portion. This is called "delayed dispatching".*

The importance will become clear in Section 6.4, because it impacts on the software adaptation. Certain tasks are triggered by the occurrence of a TIC, which is determined in the interrupt handler. These tasks are thus not executed at the TIC, but as soon as possible afterwards.

Adding a task is described in [20] and basically involves adding a TCB entry which includes the entry function and stack size for the task, and starting it in `startup.c`.

Messages

A convenient way to send data between tasks is by using messages. This is a function of the OS and system calls are used to send and retrieve messages. The maximum size of the message and the message buffers size needs to be specified, and each message buffer has a unique ID.

The reason why messages are important to us is that they can be used as triggers. A task can wait on a message, only executing when that message has been received. E.g. the display task (which handles the output messages) should run after the navigation task, outputting the solution after it has been calculated. For this messages are used.

Some system calls for utilising messages are needed:

- `i_cre_mbf(Message Buffer ID, Buffer Attributed)`: This call creates the message buffer in memory with its unique ID and attributes such as maximum message size and message buffer size
- `i_psnd_mbf(Message Buffer ID,Message,Message size)`: Sends the message.
- `i_rcv_mbf(Message, Message Size, Message ID)`: Waits for the message receipt before continuing with the task.

If a task is waiting on a message, the task that issued the system call will wait on a receive message wait queue until the message arrives.

To add a message, a unique message identifier needs to be defined, the max message size and message buffer size has to be defined in `startup.c`, and the buffer created.

6.4 Operating System Adaptations

6.4.1 ARM Project Structure

The directory structure is inherited from the SIGNAV project structure. All the source code is in the `\src` directory, with `\dsp` containing the GPS source code, `\itron` containing the RTOS source code, `\scatter` the memory scatter map (see Section 7.7.1) and `startup.c` in the `\src` directory itself. `debug` is the output directory for the object files, the ARM Executable Format (AXF) image and the binary image. The specification for the RTOS is in `\tronspec`, the required libraries in `\lib` and the project files and memory map output in the project parent directory:

```
Project Directory   [project files, memory map]
|-- debug           [object files, images]
|-- lib             [ARM libraries]
|-- src             [source parent, startup.c]
   |-- dsp          [GPS source]
   |-- itron        [RTOS source]
   |-- scatter      [Memory Scatter Map]
|-- tronspec        [RTOS Specification]
```

The `fromelf` command is used to convert the AXF file to binary format for downloading to the receiver.

6.4.2 Critical Sections

The original Architect software uses a simple variable to disable task switching during critical sections (see Section 6.2). In the SIGNAV implementation, critical sections are used. Any code deemed to be a critical section is embraced by the `BEGIN_CRITICAL_SECTION` and `END_CRITICAL_SECTION` defines. These defines have been implemented as follows:

[cpu.h]

```
#define BEGIN_CRITICAL_SECTION
  {
  INT spsr_tmp = disint();

#define END_CRITICAL_SECTION
  enaint(spsr_tmp);
  dispatch();
  }
#define DISABLE_INTERRUPT
  disint();
```


Thus, inside a critical section, any interrupts are disabled. When the critical section is exited, the `dispatch()` function is called, which switches the task if necessary. Note the braces included in the define. If there is not an `END_CRITICAL_SECTION` for the corresponding `BEGIN_CRITICAL_SECTION`, a compiler error will be generated.

6.4.3 Serial Communications

The RS232 serial communications is used as implemented in the SIGNAV software. In the original Architect, the low-level serial receive and transmit was handled in the Interrupt Service Routine (ISR), and checks for commands in the Allocate (TAlloc) task. This was removed in the adaptation and instead the SIGNAV serial implementation was used. The reasons for this were:

- Because the Architect tasks run on the new RTOS, it was deemed best to keep the SIGNAV serial implementation, as this was written for the μITRON OS.
- By using the SIGNAV implementation, the RS232 Task is a separate task that simplifies debugging

For receiving command messages, the command task was adapted. The Command Task waits upon a message from the RS232 task, indicating that a serial message has been sent. The Command Task then has to strip the command from the message and call a routine that will process the command, `ProcessCommand()`. As in the Architect software, the command messages start and end with a STX (0x0010) and ETX (0x0011) character. These delimiters are removed and the checksum is validated. If the command is valid, a `"CMDOK"` reply is sent, or otherwise a `"CMDERR"` reply. The Command Task is listed in Appendix C.2.

For the report outputs, the `SendString()` function was rewritten to adapt it to the SIGNAV RS232 task and its operation. The message format can either be NMEA or the proprietary format used in the Architect. For strings exceeding 200 characters, the string has been split into parts and sent to the RS232 task in separate messages. The first character in the message is a dummy character, as it specifies the port the output should be sent through and as in the command messages, the output message is started and ended with an STX and ETX character. The `SendString()` function is listed in Appendix C.2.1.

6.5 Adaptations required by hardware changes

6.5.1 Correlator

In [28] it is shown that the correlator subsystem on the GP4020 is identical to the correlator block found on the GP2021, but the base addresses of the registers have changed. For the GP4020, the Base Address for the 12-channel Correlator block is 0x4010 0000; for the GP2021 it is 0x00200000. The RTC, DUART, System and General control register addresses have been removed from `corraddr.h`, as the SIGNAV method is used instead of the Architect method to access these registers. Other than that, only the `BASE_CORRELATOR` define have been altered from

```
#define BASE_CORRELATOR      0x00200000
```

to

```
#define BASE_CORRELATOR     0x40100000
```

The SYSTEM_SETUP register has to be set up to the required values. Table 6.1 lists the register description with the used bits. For a full description see [28]. As in the Architect code, the MEAS_INT_SOURCE bit is set to zero, as the TIC interrupt is cleared in the ISR which examines the ACCUM_STATUS_B register. The interrupt period is not of concern for the system setup, as the value for the interrupt is set manually. Bit 5 is of concern when disabling and enabling the interrupts with the `disable()` and `enable()` functions in `io.c` (see Section 6.2).

Table 6.1: *Correlator SYSTEM_SETUP Register*

Bit No.	Mnemonic	Description	R/W
15:11	*Not Used*	...	...
10	MEAS_INT_SOURCE	'1' = MEAS_INT output cleared by a read of MEAS_STATUS_A register. '0' = MEAS_INT output cleared by a read of ACCUM_STATUS_B register.	W
9:8	*Not Used*		
7	INTERRUPT_PERIOD	...	...
6	*Reserved*		
5	INTERRUPT_ENABLE	Enables and disables correlator-sourced ACCUM_INT and MEAS_INT interrupt signals. '1' enable correlator interrupts. '0' disables correlator interrupts	W
4:0	...	...	...

The `MASK_INTERRUPT` define is set to 0x0000, and `UNMASK_INTERRUPT` to 0x0020 in `defines.h`.

6.5.2 Real-time Clock (RTC)

The GP2021 IC contains a 24-bit Real-time clock (RTC) and the Architect software made use of this. On the GP4020 there is also a RTC, but if differs in implementation

and register addresses. In the Architect the `ResetRTC()` and `ReadRTC()` functions had to be modified where `ResetRTC()` resets the RTC to zero, and `ReadRTC` reads the current value. The 24-bit value is split across 3 registers, `RTC_LS, RTC_2ND` and `RTC_MS` All three register are latched on a read of `RTC_LS`, so this register has to be read first [27, p. 43]. The maximum value of the count is $2^{24}/24/3600 = 194$ days. Upon reaching this count, the count is frozen (i.e. all ones) until a reset of the RTC.

The RTC on the GP4020 has four 16-bit registers [28, p.132], with base address 0x4010 1000 (see Table 6.2).

Table 6.2: *Real Time Clock Register Map*

Address Offset	**Register**	**Direction**	**Function**
0x000	`RTC_PRE`	Read/Write	Read Pre-scaler divider value
0x002	`RTC_SEC_B`	Read	Read Least significant 16-bits of Real Time Clock 24- bit second counter
0x004	`COMP_RTCP`	Read/Write	Comparison value for Real Time Clock Pre-scaler
0x006	`COMPS_RTCS`	Read/Write	Comparison value for 8-bits of Real Time Clock second counter & Read only access to Most significant 8-bits of Real Time Clock 24-bit second counter

The register addresses of the RTC are defined in GP4020.h as follows:

[gp4020.h]

```
typedef struct GP4020RTC
  {
  unsigned short    RtcPre;   /* Pre-scaler divider value (R/W)    */
  unsigned short    RtcSecB;  /* Bottom 16 bits of 24-bit count    */
  unsigned short    CompRtcP;   /* RTC Prescaler comparison value    */
  unsigned short    CompRtcs;   /* MS 8 of 24 bits/8-bit RTC seconds */
  } tGP4020Rtc;
#define pRTC            ((volatile tGP4020Rtc*)(pBaseAddr+pPAddrOS))
```

The only way to reset the RTC is to write a '0' to bit 0 of register `RTC_PRE` and the `ResetRTC()` function has been rewritten as:

[clock.c]

```
void ResetRTC(void)
  {
  short temp;
  pRTC->RtcPre = 0x0000;
  temp = *pTIC_RET;
  temp = (temp & 0x00ff) | ((DATA_VALIDITY_CHECK)<<8);
  *pTIC_RET = temp;
  }
```

Setting the valid value for the Data Retention register has been included in the ResetRTC() function. The Data Retention register maintains its value during ANY kind of reset. The value in the register is only altered during a complete power failure.

To read the current value of the RTC, the 16 least significant bits of the 24-bit counter are read from the `RTC_SEC_B` register. On the read, the values in register `RTC_SEC_T` will be latched. `RTC_SEC_T` is the most significant 8 bits of the `RTC_COMP_RTCS` register, and contains the 8 Most Significant Bits (MSBs) of the 24-bit RTC counter. Thus 24-bit counter value is `[RTC_COMP_RTCS[15:8]:RTC_SEC_B[15:0]]` or `[RTC_SEC_T[7:0]:RTC_SEC_B[15:0]]`.

[clock.c]

```
int ReadRTC(void)
  {
  int   Rtc24Bit;
  Rtc24Bit = pRTC->RtcSecB;
  Rtc24Bit |= ((pRTC->CompRtcs & 0xff00) << 8);
  return Rtc24Bit;
  }
```

The `ResetRTC()` and `ReadRTC()` functions calls in the Architect tasks have subsequently been maintained. The RTC of the GP4020 has the same limitation as in the GP2021 of a maximum day count of 194 days (see Section 6.5.2).

6.5.3 Non-volatile Memory (NVM)

The Non-volatile Memory (NVM) area is used to store data that needs to be retained when the receiver is reset, or even switched off. In the Architect software, a block of the normal SRAM was designated as non-volatile [1]; however, in the SIGNAV implementation the 8 kB of internal SRAM is used for this task and the backup-battery ensures that the data is retained even when no power is applied. On a satellite the backup battery should be replaced by a power supply that ensures that the NVM is continually powered when the receiver is switched off. The internal SRAM is used for NVM as in the SIGNAV implementation, and the Architect code was adapted to utilise the internal SRAM memory block. This ensures that the error-checking improvement of the Architect code is retained. A parity bit is stored for each byte written and an integrity byte is written at the start of the block.

[28] shows that the internal SRAM is located at base address 0x6000 0000; the top is thus at 0x6000 0000 + 0x1FFF = 0x6000 1FFF, whereas in the Architect the NVM top is at address 0x0017FFF. Because of the 32-bit address bus, the memory addresses are on 4-byte boundaries (the internal ram is organised as 2k x 32bit SRAM). The memory map for the Architect implementation and the new modified values are listed in Table 6.3.

The total memory used in the internal SRAM is (0x0017FFFF-0x0017ECB0+1) = 4944 bytes. Some extra space in the NVM has to be allocated for the Space Adaptation - refer to Section 7.7.2 for details.

6.6 Conclusion

The port of the GPS Architect software from the ORION hardware to the SIGNAV receiver did not only require changes because of the new hardware platform, but the tasks were also ported to a new RTOS. Both these changes have been implemented successfully and the GPS Architect terrestrial software functions correctly.

The next chapter describes the changes necessary for space use.

[1] Private Communication, Dr. Möntenbruck (DLR-GSOC)

Table 6.3: *Non-volatile memory map*

Data Set	Architect Start Address	Architect End Address	New Start Address	New End Address	Bytes
RAM Top	-	0x0017FFFC	-	0x60001FFC	-
Time	0x0017FFE0	0x0017FFF2	0x60001FE0	0x60001FF2	19
Position	0x0017FFC0	0x0017FFDB	0x60001FC0	0x60001FDB	28
Almanac (PRN32)	0x0017FF90	0x0017FFBD	0x60001F90	0x60001FBD	46
...	...	...	...	...	...
Almanac (PRN01)	0x0017F9C0	0x0017F9ED	0x600019C0	0x600019ED	46
Ephemeris (PRN32)	0x0017F95C	0x0017F9BF	0x6000195C	0x600019BF	100
...	...	...	...	...	...
Ephemeris (PRN01)	0x0017ED40	0x0017EDA3	0x60000D40	0x60000DA3	100
Rs Clock Offset	0x0017ED30	0x0017ED39	0x60000D30	0x60000D39	10
Iono/UTC Model	0x0017ECB0	0x0017ED25	0x60000CB0	0x60000D25	118
NVM Start	0x0017E001	-	0x60000001	-	-

Chapter 7

Adaptation for use in space

7.1 Introduction

In collaboration with Deutsches Zentrum fur Luft-und Raumfahrt - German Space Operations Centre (DLR-GSOC), the ported GPS Architect software was adapted to enable the receiver to work in space. The major changes to the software were:

- Changing the operational parameters, e.g. the ITAR velocity and altitude restrictions have to be altered for LEO operation (Section 7.2).
- Adding an orbit propagator to the software to guide the signal acquisition process. The propagator is based on the SGP4 orbital model coupled with TLE data to predict the receiver position and velocity for Doppler predictions. (Section 7.3).
- Adding output messages that are more suitable to LEO operation. They are highly compatible with DLR's SGPS receiver as a result of the collaboration (Section 7.4).
- The spherical navigation solution was replaced with a cartesian solution. The cartesian solution is simpler and avoids solution singularities at the poles, which can occur on a satellite with a polar orbit. (Section 7.5).
- Addition of Pulse-per-Second Synchronisation of the navigation solution. This change ensures that the navigation solution is computed at the integer GPS second which is extremely useful when doing simulations. (Section 7.6).
- Adapting the software to the extra memory requirements of storing the TLE data in non-volatile memory and the stack for the SGP4 Task. (Section 7.7).

7.2 Operational Parameters

The ITAR limitations have been removed to operate in LEO conditions, and depending on the receiver mode, either the ITAR limits or the new limits are used. The maximum receiver velocity used for fast syncing (`MAX_RECEIVER_VEL`) has been redefined.

[define.h]

```
#define SPEED_LIMIT_LEO  25000.0     /* metres per second. */
#define HEIGHT_LIMIT_LEO 10000000.0  /* meteors. */
#define MAX_RECEIVER_VEL 8000        /* metres per second. */
```

The elevation mask has been augmented with a mask used in the navigation solution, `NavElvMask`. The elevation mask determines which satellites are tracked in the available channels. The navigation mask determines which satellites are actually used in the navigation solution. With the change, SVs with low elevations that have just been acquired are not included in the solution as the pseudorange measurements have been found to be inaccurate right after lock. The navigation mask is set to 5° degrees above the elevation mask.

[initarch.c]

```
ElvMask = 0.0f;    /* The default elevation mask. */
NavElvMask = 5.0f;
PdopMask = 99.0f;  /* The default PDOP mask. */
```

[nav.c]

```
if((((observed->satmap)&(1UL<<sat))==0) && (ephs[sat].vflg==VALID) &&
  (ephs[sat].s1hlth==0) &&
  !(ielvd[sat]<NavElvMask && TrackMode!=COLD_START))
{
  ...
```

In the original Architect, only the SV velocity is used to predict the Doppler frequency offset. This has been altered to include the receiver velocity as well.

7.3 Orbit Propagator

7.3.1 Operation

To aid the Doppler prediction, a task which calculates the estimated position of the receiver was added. The SGP4 Task uses TLE values that are uploaded to the receiver. Then the SGP4 solution is used in the channel allocation process to estimate the Doppler offsets. Usually, if the receiver is operating in Highest Elevation mode (see Section 2.5) the last known fix is used, together with the almanac, to predict the receiver position. However, the SGP4 solution is used to "force" the receiver into the Highest Elevation Mode. Thus, the **Almanac is required** for LEO operation during start-up.

For LEO use the sequence of operations is:

- Set the current date and time from the On-Board Computer (OBC) of the satellite
- Upload a recent GPS Almanac and TLE data (See Section 7.3.2)
- Force the receiver into Highest Elevation Mode

The SGP4 algorithm requires a transformation from ECI to ECEF coordinates as set out in Section 2.3.3. This is done in the SGP4 Task. The angle is calculated in the `GST()` function and the coordinate transformation in the `ECItoECEF()` function. These functions are listed in Appendix C.1.

7.3.2 Almanac and TLE uploads

The Almanac and TLE files have to be uploaded to the receiver for operation. This can be done from the ground segment and the data size is minimal (<20kB). After launch, if TLE data are not available yet, estimated orbital elements can be used. The aim of the propagator is not to determine **exactly** what the Doppler offsets are, but only to guide the search. Even if a very precise prediction is available, the receiver clock offset degrades the estimate and some frequency bins will always have to searched. As soon as a fix is obtained and satellite visibility is kept above a minimum, the propagator is not required.

If possible, testing should be conducted for a Cold-Start scenario (See 9.2) to see whether the signals can be obtained in this mode. Because of the high Doppler shifts, cold start times can be up to 13 times longer (5 kHz vs. 65 kHz) than for the terrestrial case.

7.3.3 Propagator and orbit manoeuvres

If enough satellites are not visible continually, the receiver will lose lock and Doppler prediction may start playing a role again. If, for example the antenna is mounted on the -Z side of the satellite and the +Z axis is pointing toward nadir most of the time, orbit manoeuvres could point the antenna away from the constellation and lock will be lost.

If lock is lost for a short space of time, the last known navigation solution is used for Doppler prediction and rapid acquisition. However, because the satellite is moving at such a high speed, even a short loss of lock can result in difficulty reacquiring the satellites. Then the propagator will have to be employed again. Of course the ephemeris will still have to be valid (less than four hours old) for a rapid navigation solution[1].

7.3.4 Memory and processor load

Using the μTRON RTOS gives us the ability to determine the stack size and processing time that each task uses, which was not available in the Architect. The exact processing burden of each task can be seen.

The SGP4 task executes every 2 seconds. At this rate, it uses a maximum of 2.9% of processing time, an average of 2.8% and is allocated a stack size of 0x800 bytes. During operation 1458 bytes of the task stack is still free.

7.4 Output Messages

The original Architect messages are inadequate for displaying information for receiver operation in LEO circumstances:

- The default longitude, latitude and height display is replaced with coordinates in ECEF cartesian coordinates.
- The original channel status and satellite summary output message does not accommodate the large Doppler frequency shifts: too few character are available to display the large values.

[1]If the ephemeris data are valid, the navigation solution can be computed as soon as the satellite signal is locked onto — the 30 second delay to download the data is avoided.

- Output of the current TLE values used in the SGP4 propagator was added

In the original Architect, the display task was set up to output all the sentences applicable to receiver operation (in essence ALL of them) depending on whether in NMEA or WINMON (Proprietary Zarlink output) mode. This has been changed to a programmable scheme. The sentences that are output as well as their period are programmable. This functionality can be used to minimise the load of the Display Task on the processor. E.g. the satellite summary sentence (F04) which contains the visible satellite information can be output every 10 seconds as satellite visibility changes slowly. Also, the channel status message (F03 in Terrestrial Mode or F43 in LEO Mode) is useful, but not necessary. The most important message is the position report in ECEF coordinates (the F40 sentence in LEO Mode) which should normally be output every second.

The command to program the output sentences is the "DR" command (See Appendix D.1.2), which selects the output period and the frame number. A programmed rate of "-1" sets the sentence off, "0" outputs the sentence once and positive integers indicate the sentence output period. The most important output sentences are listed in D.2, the full WINMON messages can be found in [12].

7.5 Navigation Solution

By default, the Architect navigation solution is in spherical coordinates. This method is more computationally intensive than computing the solution in cartesian coordinates. When using the spherical method it gives the opportunity to "solve" for position and velocity with only 3 satellites visible; however, the solution assumes that the altitude is constant from the last fix. When using spherical coordinates, the equations can be linearised around azimuth, elevation and radial distance (or altitude). It seems that the Architect navigation solution has a bug [2] and the solution was rewritten to use cartesian coordinates, implying that a minimum of 4 satellites are required for a fix. The cartesian navigation algorithm is discussed Section 2.7.

The execution of the Navigation Task is TIC dependent (see Section7.6). The Nav Task waits on a message from the ISR for execution so that the Nav Task will execute on the measurements taken on the integer second. The Display Task waits on a message from

[2]Private Communication, Dr. Oliver Möntenbruck, DLR-GSOC

the Nav Task, so the displayed solution will be the most recent position/velocity fix. Both these tasks execute nominally once a second.

7.6 PPS synchronisation

The reference values from the simulator during testing are values for each integer second. If the receiver values are navigation solutions on each integer second then direct comparison to the reference values is possible. Thus the aim is to synchronise the time at which the values used for the navigation is obtained (a TIC) with the integer second of GPS time. This is called Pulse-per-Second Synchronisation (PPS).

If a pulse is generated on each integer second, it can be used by other hardware subsystems as a timing reference as GPS time is synchronised to UTC to within 1 μsecond (see Section 2.2.2).

The period resolution of the TIC is 175ns. This is derived from the 40 MHz clock period divided by seven [28]. The nominal TIC period is 0.099 999 9 seconds. The aim of PPS syncing is to keep every tenth TIC (the TIC that the Navigation solution occurs on) aligned to the integer GPS second.

Thus the PPS algorithm should:

- Align every tenth TIC to the integer second as closely as possible
- Align the navigation solution task to the TIC which occurs on the integer second

As soon as a navigation solution is computed (which includes the clock solution), the TIC period can be adjusted so that every tenth TIC occurs on the integer second and the navigation task should be triggered by that TIC.

To illustrate PPS, the example in Figure 7.1 is used. The time of the navigation solution is at x.715 seconds. Thus the TIC is either 0.085 seconds early, or 0.015 seconds late; the navigation solution is either 0.285 seconds early, or 0.715 seconds late. To align the Navigation TIC to the integer second, there are options:

- Set the new TIC period to 0.085s, set the Nav solution to occur in 3 or 13 TICs
- Set the new TIC period to 0.185s, set the Nav solution to occur in 2 or 12 TICs

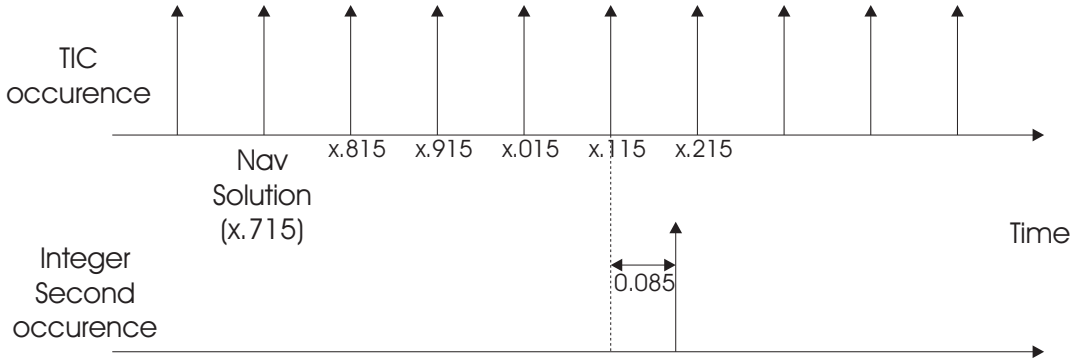


Figure 7.1: *PPS Synchronisation*

Depending on the time at which the synchronisation occurs, different options warrant different solutions. If the TIC is delayed for too short a time, the ISR may not be able to finish or measurements may not be taken completely before the next interrupt. To be on the safe side, it is advisable to adjust the TIC period to the nominal period plus the offset, so the TIC period will always be between 0.099 999 9 and 0.019 9999 9 seconds.

A big shift in the TIC period is expected to occur only once when the clock solution is obtained on the condition that tracking is maintained. Once lock has been lost, the receiver clock may drift and when a solution is available again, the TIC is realigned. If lock is maintained, the only adjustment necessary is the 1ppm resolution error and the receiver oscillator error.

7.7 Memory

7.7.1 Memory Map

The ORION hardware design uses an ARM60B 32-bit processor, and the GP4020 has an integrated ARM7TDMI core. The latter gives the possibility to compile the software in Thumb Mode, which yield smaller code sizes [22]. The Thumb functionality has been implemented and the code size has been improved from the Architect/ARM60B implementation. In the SIGNAV implementation, routines that do not need short execution times are run from Flash, and the high-speed routines are loaded into the SRAM. The software that was developed is small enough so that ALL the object code is loaded into the SRAM for execution.

The text file that describes which object files are places where is described in

"SCATDES3.txt". This file is located in the \src\scatter directory. The file is inherited from the SIGNAV project structure.

7.7.2 Non-volatile Memory

For storing the TLE values used in the propagator, extra space in the non-volatile memory (NVM) has to be allocated. A parity byte is stored for each eight data bytes and if necessary, zero padding bytes are added to make up the 8-byte block. An integrity byte is stored at the beginning of each block to guard against corrupted data. The parity bit is important to detect data corrupted by single-event upsets.

The TLE structure is 128 bytes, so the total memory required is 1+ceil(128/8)*9 = 145 bytes. The TLE read and write functions are:

[nvm.h]

```
#define NVM_TLE                    0x60000C18UL
```

[nvm.c]

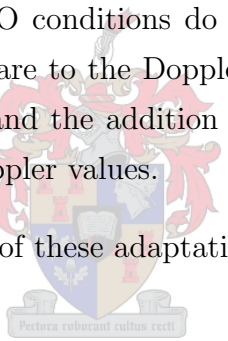
```
void NVMStoreTLE(void)
{
  char *ptr;
  tlestruct kep;
  /* Get a pointer to where the NVM time is stored. */
  ptr = (char *)(NVM_TLE);
  /* Get the current TLE values time. */
  BEGIN_CRITICAL_SECTION;
  kep = currentkep;
  END_CRITICAL_SECTION;
  /* Store the data. */
  StoreNVMData(ptr,(char *)&kep,sizeof(kep));
}

logical NVMReadTLE(tlestruct *kep)
{
  char *ptr;
```

```
  /* Get a pointer to where the NVM time is stored. */
  ptr = (char *)(NVM_TLE);
  /* Read the data. */
  if(ReadNVMData(ptr,(char *)kep,sizeof(*kep))==FALSE)
      return(FALSE);
  else
      return(TRUE);
}
```

7.8 Conclusion

Adaptation of the software for LEO conditions do not require fundamental changes in the software. The biggest changes are to the Doppler prediction algorithm and the new SGP4 task, PPS synchronisation, and the addition of output messages that can handle the large position, velocity and Doppler values.

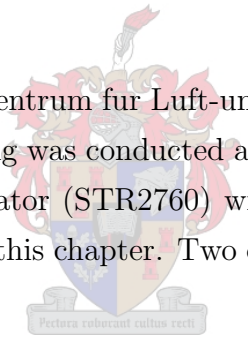
The next chapter show the success of these adaptations during LEO simulations.



Chapter 8

Simulator Results

8.1 Introduction



In collaboration with Deutsches Zentrum fur Luft-und Raumfahrt - German Space Operations Centre (DLR-GSOC) testing was conducted at their facilities in Oberpfaffenhofen, using a Spirent GPS signal simulator (STR2760) with LEO scenarios. The results of 4 testing scenarios will be shown in this chapter. Two distinct types of scenarios were used:

- Error-free scenarios
- Scenarios with ephemeris and ionospheric errors included

The error-free scenarios were run to test the raw navigation accuracy and satellite tracking performance of the receiver. The scenarios are for a satellite in a polar orbit at an altitude of 445 km. The scenario is the same used by DLR in [13] so the results could be compared. The parameters for the test satellite are:

Semi-major axis [km]	6823.0
Eccentricity	0.001
Inclination [degrees]	87.0
RAAN [degrees]	0.0
Mean anomaly at epoch [degrees]	0.0

All the results share this configuration, except that the mean anomaly at the epoch for some test cases differ by a degree or two. A plot of the orbit can be seen in Figure 8.1. The plot was generated from values logged by the receiver.

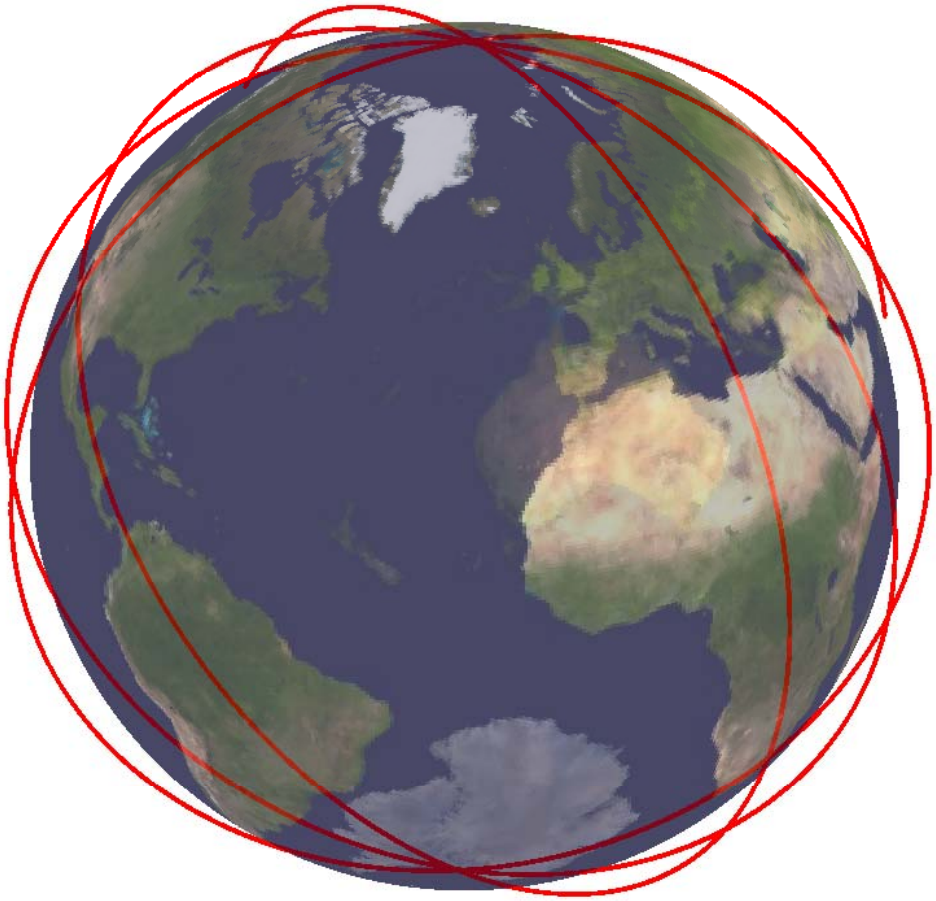


Figure 8.1: *3D Plot of simulator test scenario*

The navigation accuracy results are displayed in the radial, along track and cross track directions. This is a method that yields results that can be examined for certain errors. For example, an offset in the along-track position could indicate that the clock offset solution has an error.

To enable accurate comparison, the receiver forms a solution on every integer second (see Section 7.6) called Pulse-Per-Second Synchronisation (PPS). The reference values from the simulator are extracted for position and time at every integer second. Thus, the results do not have to be interpolated for comparison.

The results are organised into sections:

- Error-free results showing the navigation accuracy of the SGPS receiver. The results of the navigation solution are shown in Section 8.2.1 and the raw pseudorange and pseudorangerate measurements in Section 8.2.2

- Results for a simulated space-environment which include ionospheric and ephemeris errors in Section 8.3. These results indicate the accuracy expected from the receiver on an actual mission.
- Sections 8.4.1 and 8.4.2 show the results of an adaptation of DLR's code on the SIGNAV receiver. The main difference is the use of carrier phase tracking to increase measurement accuracy.

The results were successful and show raw navigation accuracy to within a few metres and that the orbit propagator enables acquisition of multiple satellites within 30 seconds.

8.2 Error-Free results

8.2.1 Navigation Accuracy

The results in this section show the navigation accuracy for an error-free LEO scenario over a period of 351 minutes.

Figure 8.2 shows the summary of results for this scenario: the number of satellites tracked, the ECEF position and ECEF velocity. Figures 8.3 and 8.4 show the position and velocity solution results compared with the reference values taken from the simulator in radial, along-track and cross-track components.

The navigation solution accuracy has mean and standard deviation error values of:

Direction	Mean [m]	Standard Deviation [m]
Position		
Radial	0.12105	1.8483
Along Track	0.21185	0.95439
Cross Track	-0.038383	0.7386
Velocity		
Radial	-0.10009	0.42301
Along Track	0.013709	0.16569
Cross Track	0.00035619	0.14943

During tests a slight problem with PPS was discovered. On three occasions during the plot shown in Figure 8.3, the integer second alignment is out by one TIC, and the resultant along-track position error shows a magnitude of approximately 760 m, which is the

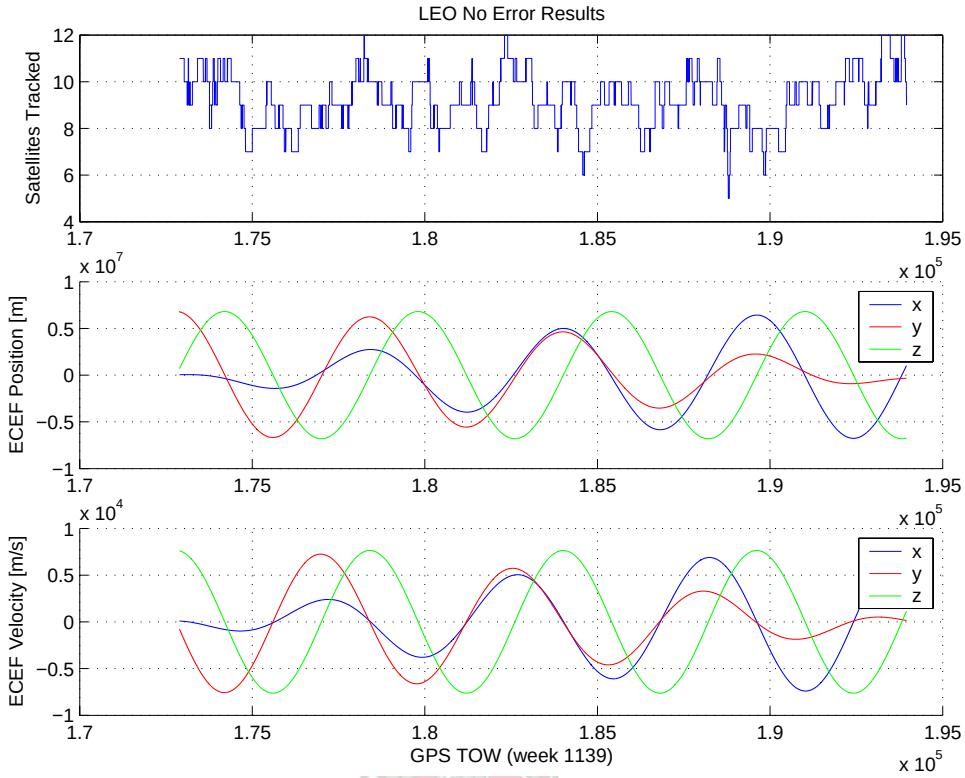


Figure 8.2: *No Error LEO Summary*

expected position error for a velocity of 7.6 km/s. The plot, including these timing errors is shown in Figure 8.5, with the instances circled in red. **Note: The solution is still correct, but the time output indicates the solution is not synchronised to the integer second.**

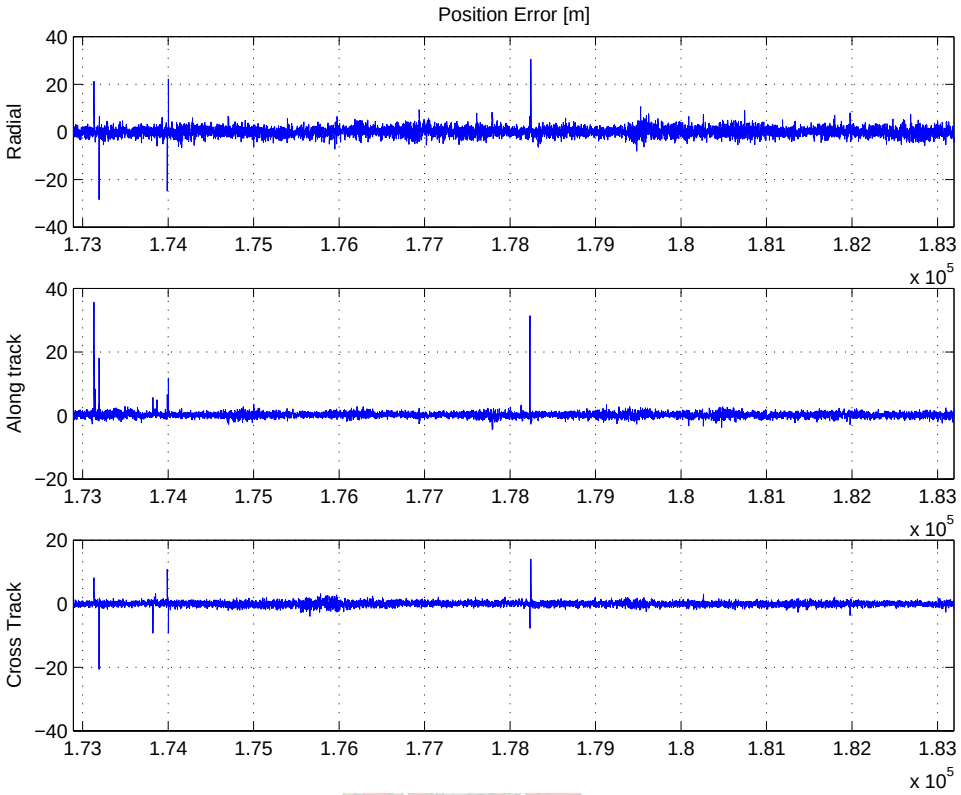


Figure 8.3: *LEO Position Navigation Accuracy*

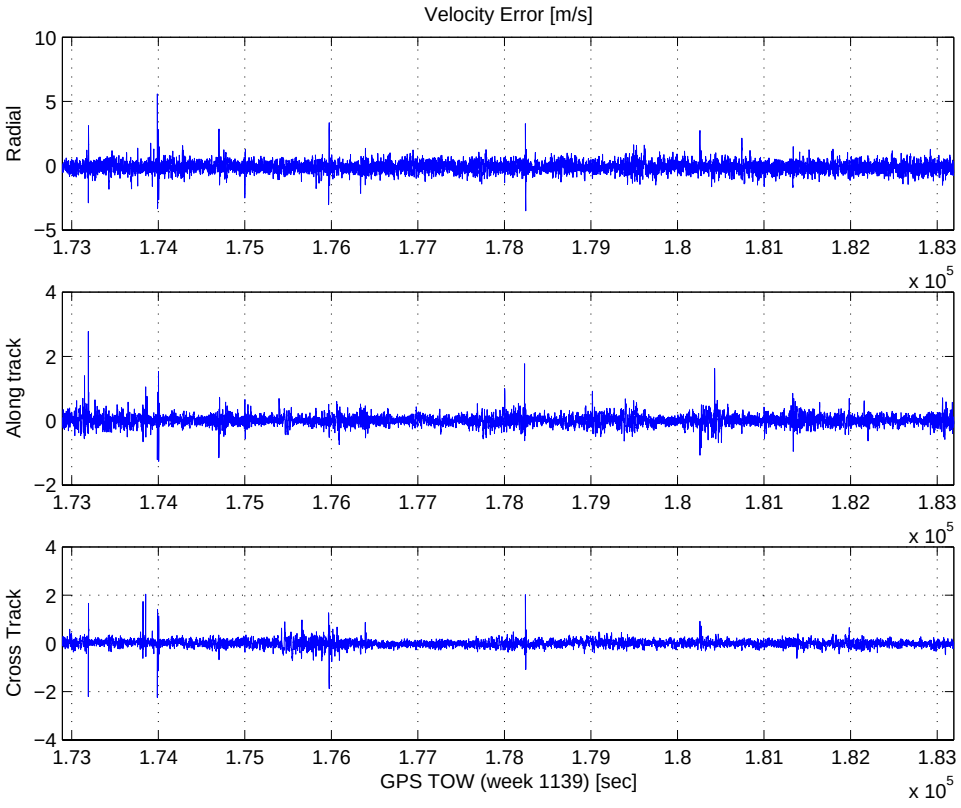


Figure 8.4: *LEO Velocity Navigation Accuracy*

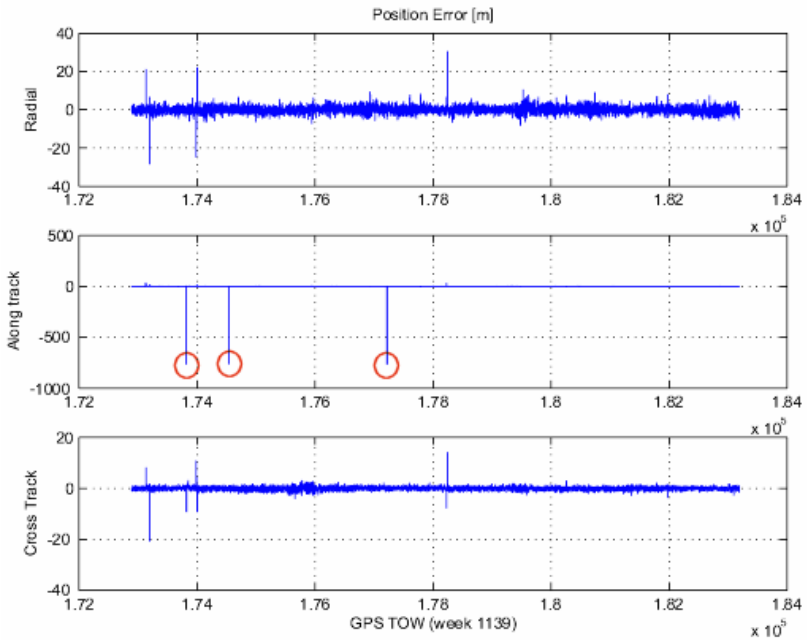


Figure 8.5: *LEO Position Accuracy with PPS Errors*

8.2.2 Raw Measurement Results

Pseudorange Measurements

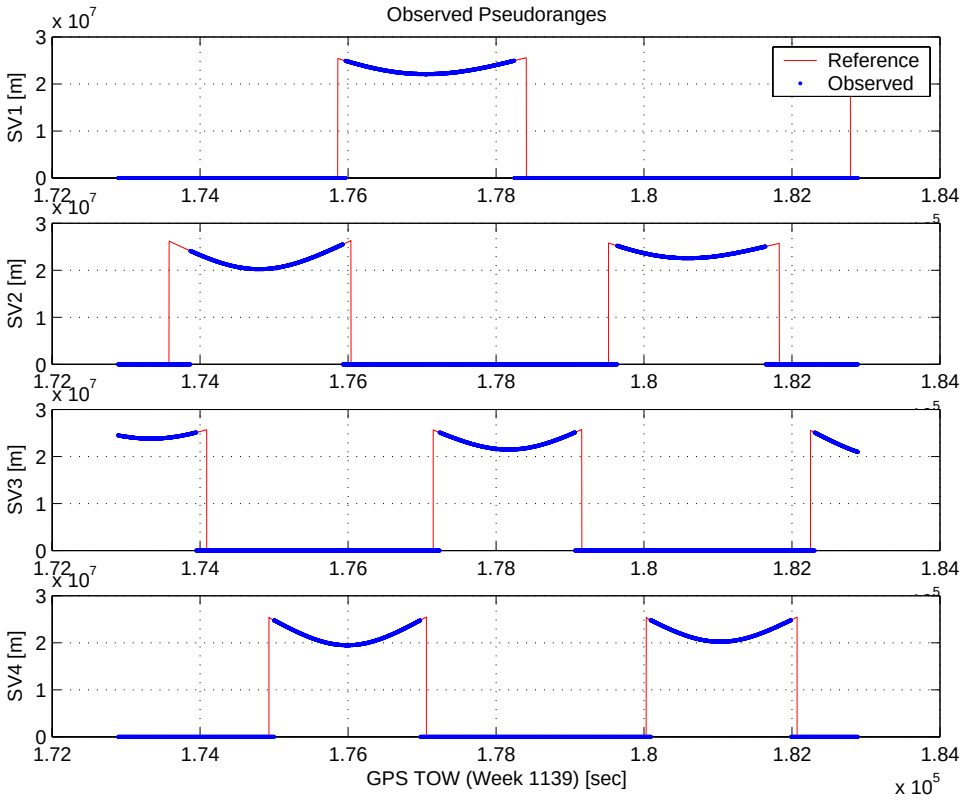


Figure 8.6: *Pseudorange Measurement*

The observed pseudorange and reference range for the first 4 SVs is plotted in Figure 8.6. The difference between the calculated and reference values is shown in Figure 8.7.

The plot of the difference shows an interesting phenomena. The error seems periodical and has a magnitude of either approximately 105 or 52 metres. At first this seems strange, but dividing by the speed of light the error translates to a possible timing error of either 175 or 350 ns. Now, remembering that the resolution of the TIC period is 175ns (see Section 7.6). This 105 or 52m offset does NOT occur in the navigation solution, which indicates that it is not an error at all, but an artefact of the comparison between the simulator reference range and the measured pseudorange.

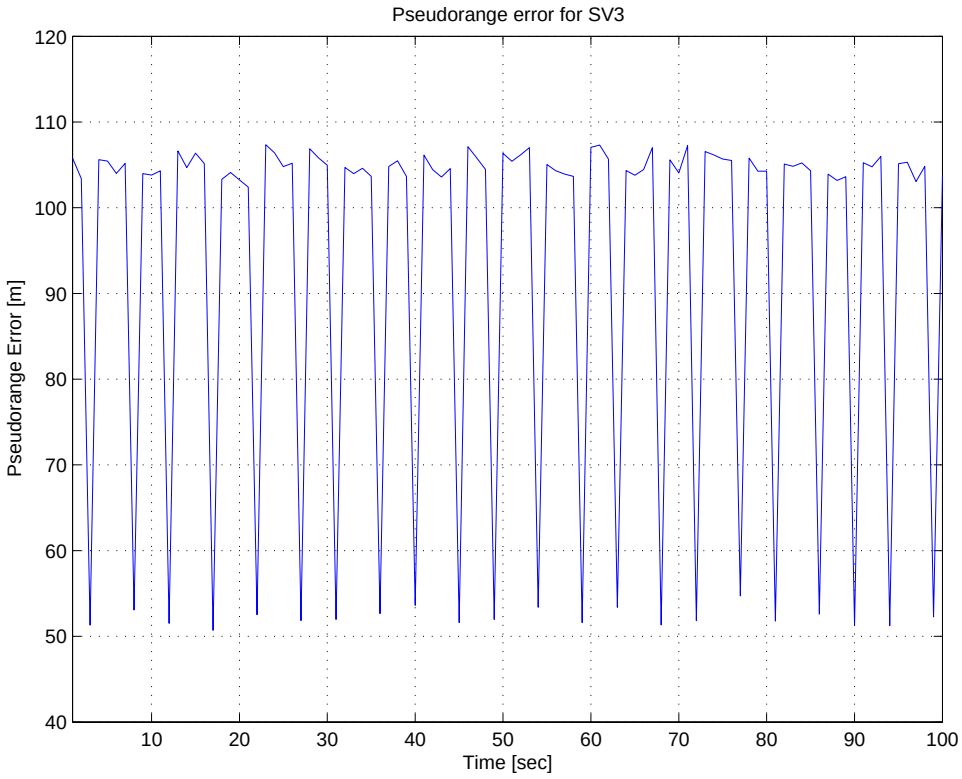


Figure 8.7: *Pseudorange Measurement Error*

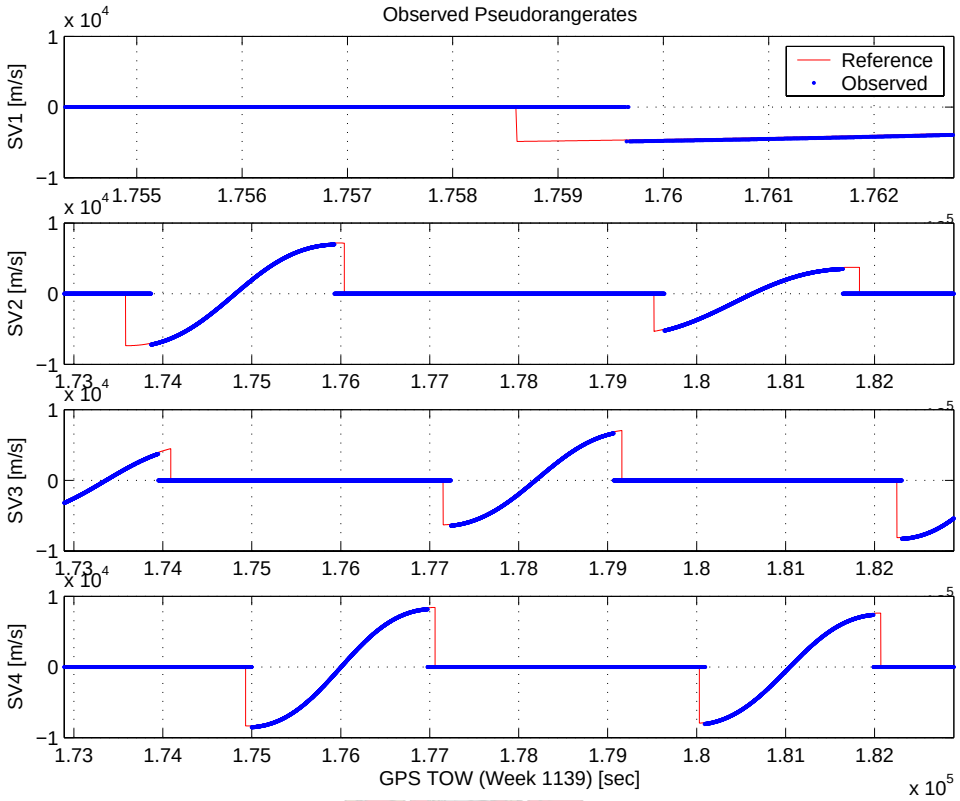


Figure 8.8: *Pseudorangerate Measurement*

Pseudorangerate Measurements

The observed pseudorangerate and reference range rate for the first 4 SVs is plotted in Figure 8.8. The difference between the calculated and reference values is shown in Figure 8.9. Note again the apparent error which does not seem to be an error because it does not occur in the velocity measurement error, but seems to be merely an artefact of the comparison process.

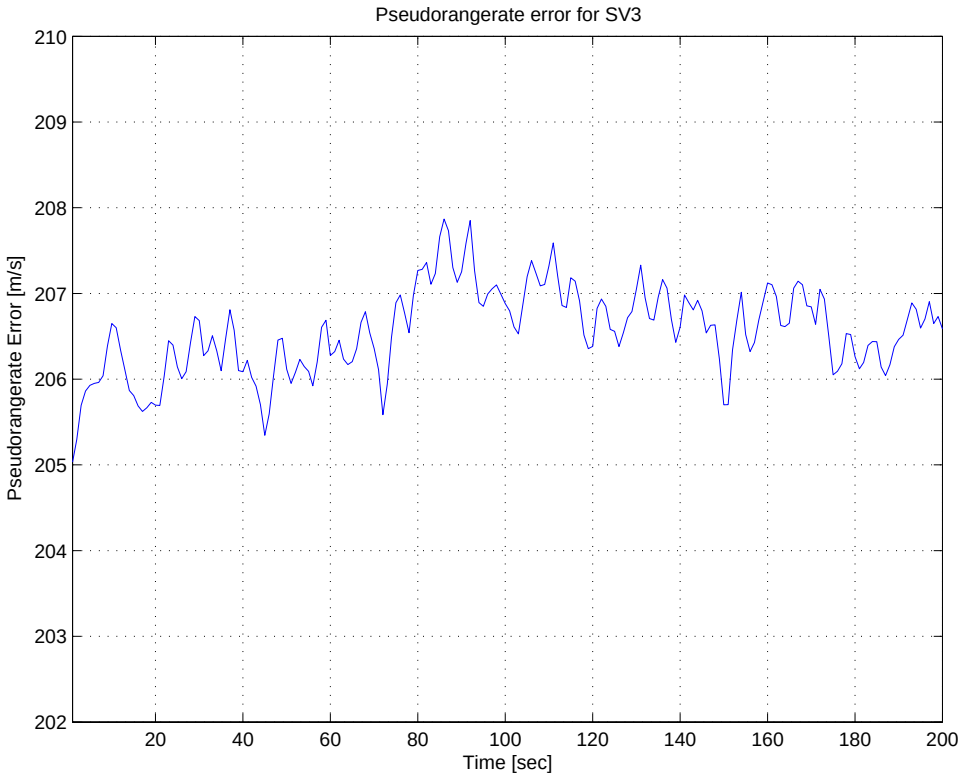


Figure 8.9: *Pseudorangerate Measurement Error*

8.3 Results with deliberate errors

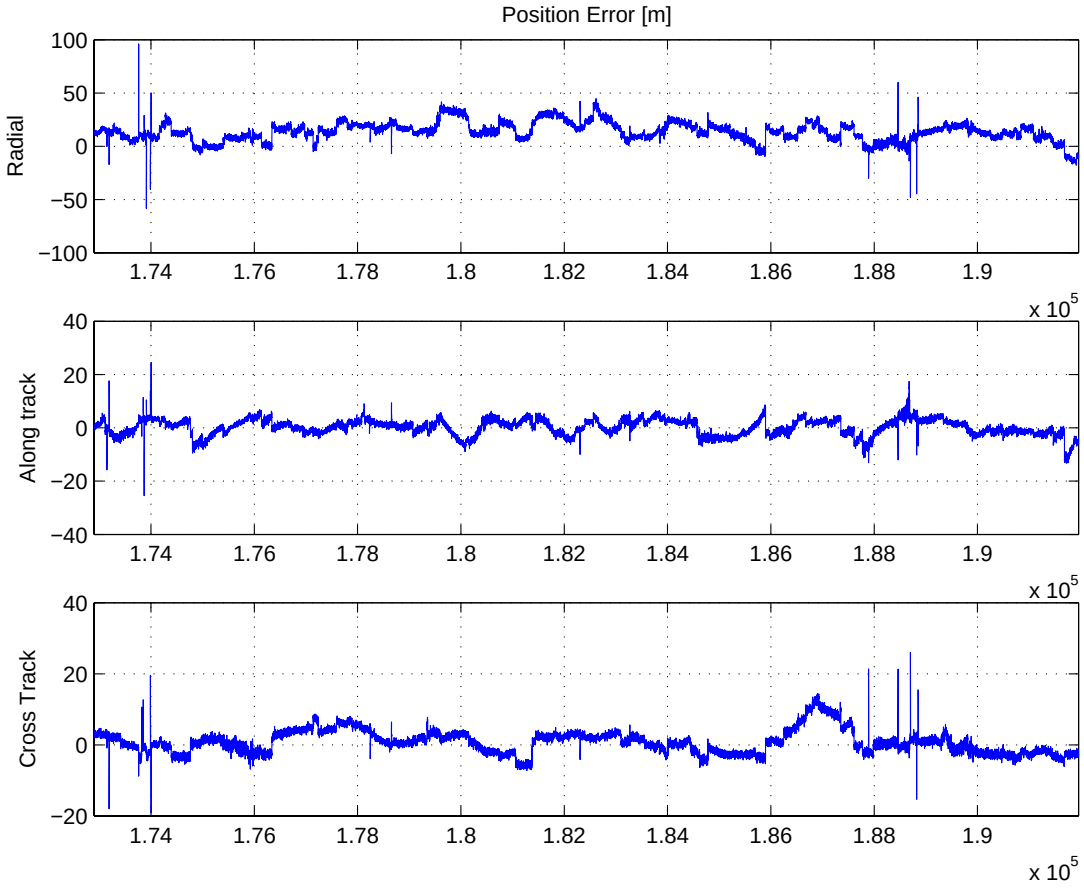


Figure 8.10: *LEO Position Navigation Accuracy with Errors*

To better simulate real-world circumstances tests were done that include ionospheric errors and ephemeris errors. The results portray the accuracy that is expected when actually flying the receiver on a satellite. The mean and standard deviation error values are:

Direction	Mean [m]	Standard Deviation [m]
Position		
Radial	13.5647	8.9747
Along Track	0.15941	3.0559
Cross Track	0.76592	3.3621
Velocity		
Radial	-0.12559	0.46739
Along Track	-0.012432	0.16817
Cross Track	0.001178	0.15502

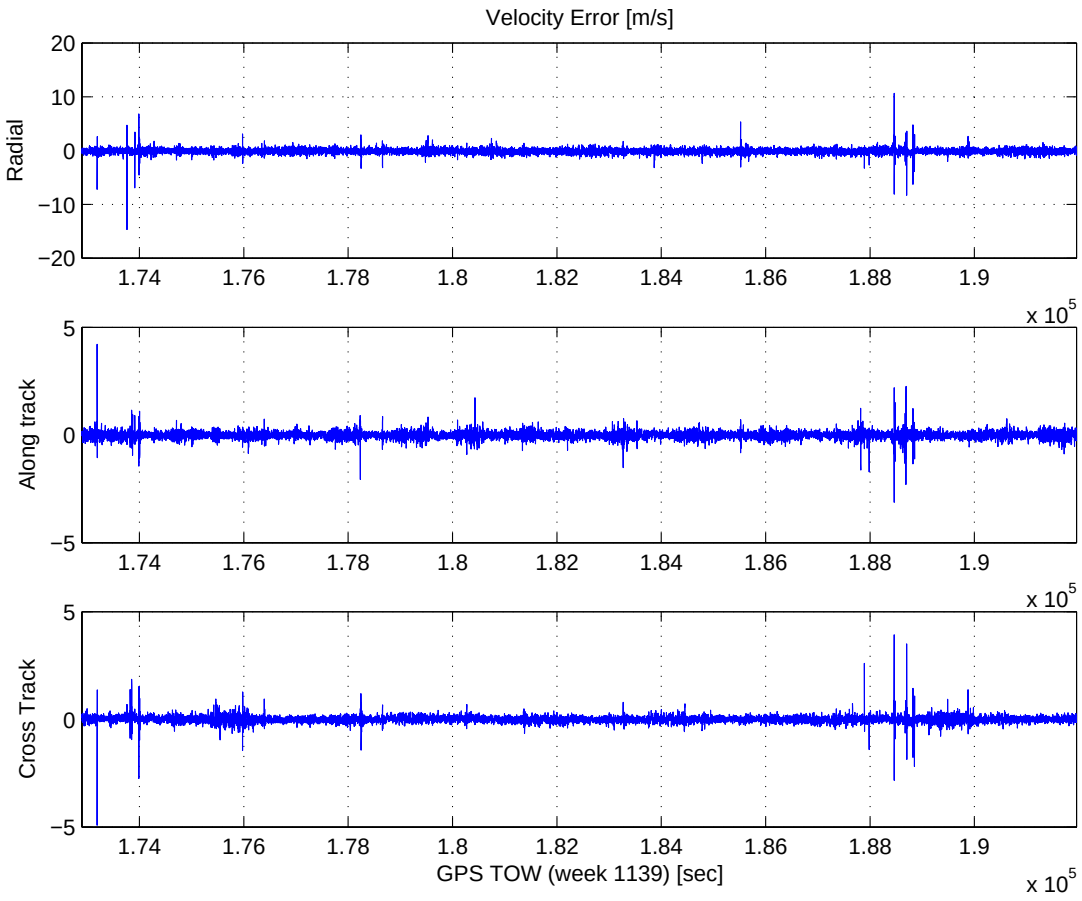


Figure 8.11: *LEO Velocity Navigation Accuracy with Errors*

8.4 DLR Carrier Phase smoothed results

8.4.1 DLR Navigation Accuracy

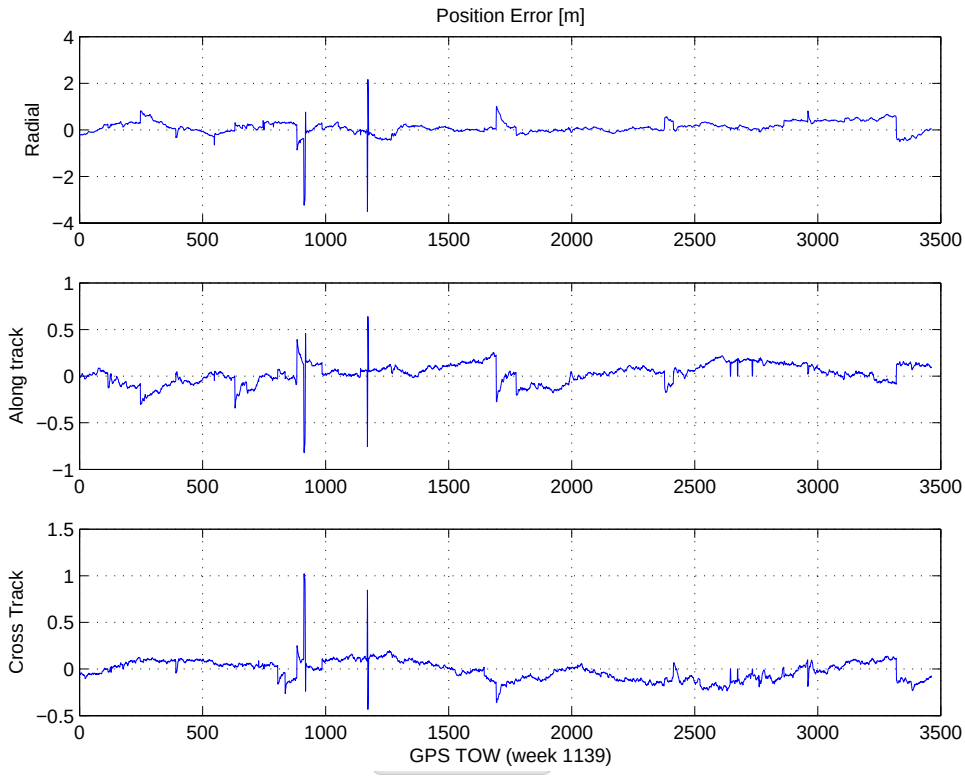


Figure 8.12: *LEO Position Navigation Accuracy (Carrier Smoothed)*

In collaboration with DLR-GSOC, the latest version of their software was tested on the SIGNAV receiver. The software include carrier phase measurements and it is used to smooth the pseudorange measurements. The results of the tests are seen in Figures 8.12 and 8.13. The navigation solution accuracy has mean and standard deviation error values of:

Direction	Mean [m]	Standard Deviation [m]
Position		
Radial	0.10088	0.28234
Along Track	0.027396	0.10824
Cross Track	-0.012831	0.10897
Velocity		
Radial	0.0056346	0.027878
Along Track	0.0026783	0.011697
Cross Track	-0.0041588	0.013576

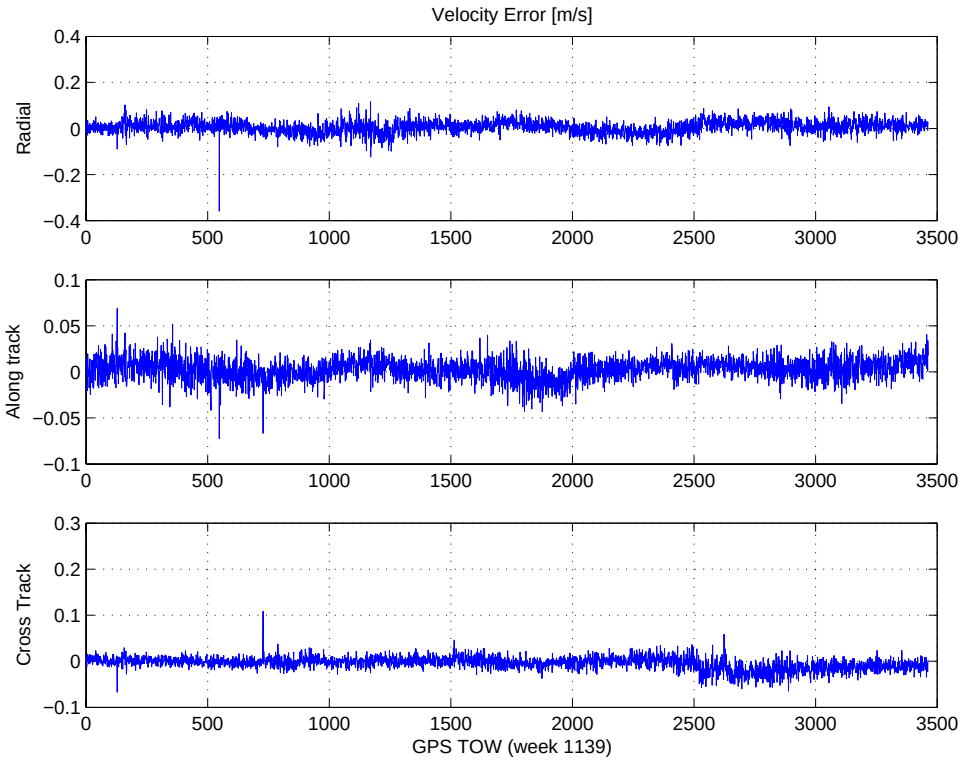


Figure 8.13: *LEO Velocity Navigation Accuracy (Carrier Smoothed)*

The most apparent difference in the navigation accuracy is the absence of noise on the position measurements, and the much greater accuracy of the velocity solution.

The scenario is the same one used for the developed software version (see Section 8.2) and the difference is notable. The navigation solution for both position and velocity does not suffer from the same amount of noise as seen previously. However, when simulations with ionospheric and ephemeris errors are compared, the difference becomes negligible because of the errors introduced by these two sources.

8.4.2 DLR Results with deliberate errors

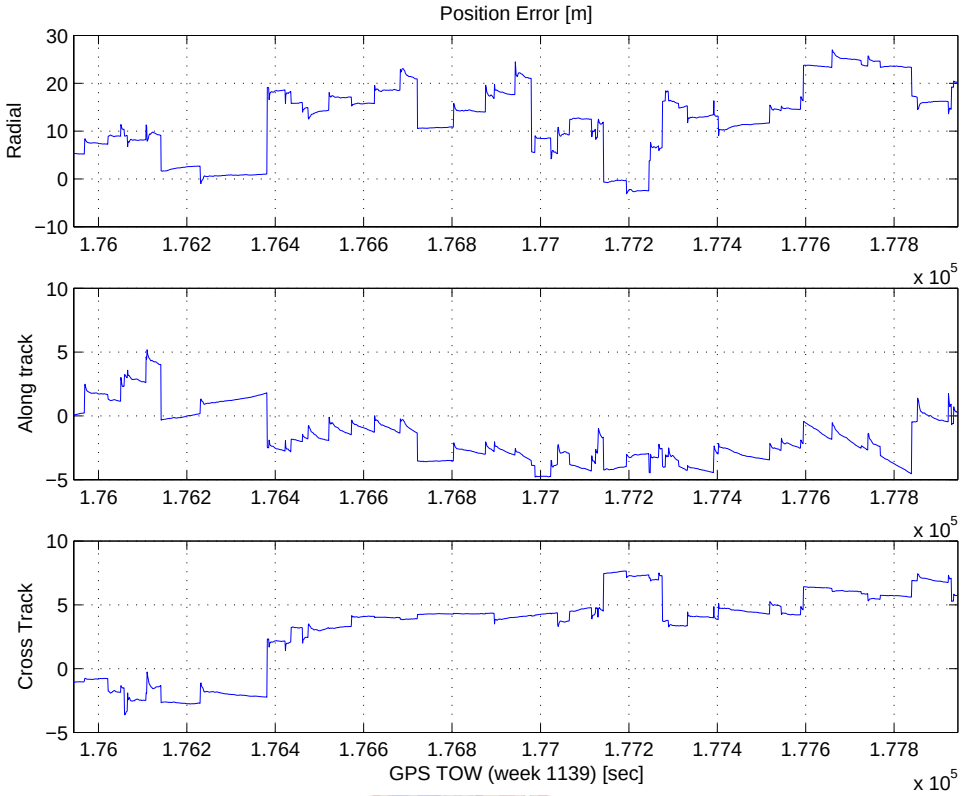


Figure 8.14: *LEO Position Navigation Accuracy with Errors (Carrier Smoothed)*

The DLR-based software on the SIGNAV receiver with ionospheric and ephemeris errors yield the following mean and standard deviation values:

Direction	Mean [m]	Standard Deviation [m]
Position		
Radial	12.5922	7.4181
Along Track	-1.6297	2.0641
Cross Track	3.2817	3.0253
Velocity		
Radial	0.0083961	0.040598
Along Track	-0.019191	0.015662
Cross Track	-0.0047744	0.012354

The DLR software was ported to the SIGNAV receiver, but with limited time available the results are not as good as can be expected and some problems were observed. Again, the carrier smoothing yields much better results for the velocity solution. The position solution accuracy is comparable to the results without carrier smoothing. The accuracy

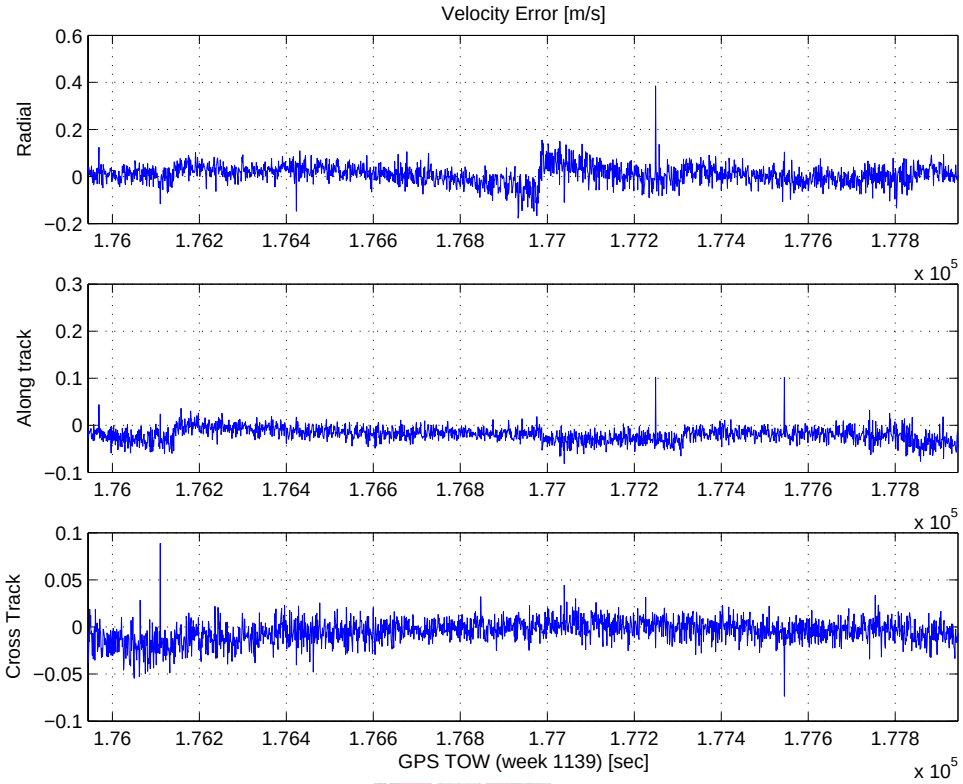


Figure 8.15: *LEO Velocity Navigation Accuracy with Errors (Carrier Smoothed)*

advantage of carrier smoothing in the position solution is lost in the ionospheric and ephemeris errors. The biggest advantage to carrier phase tracking is only applicable to dual-receiver situations, such as formation flying of two or more satellites. Then the GPS subsystem can be used for very accurate relative navigation between the satellites.

8.5 Conclusions

The tests were successful and show that the receiver is space-capable. The results show an expected navigation accuracy of $<$ 40m in the radial direction, which is the least accurate component of the navigation solution. Along and cross track errors are confined to $<$ 15m.

The mean offset in the radial solution is due to the ionospheric model used in the simulator. The altitude of the LEO test was 450km, and for satellites at higher altitudes than this the mean error will reduce. If ionospheric data is available from another source such as a dual-frequency receiver, the radial mean error can be removed in post-processing.

The results obtained by using carrier smoothing is as accurate as those without carrier smoothing, although less noisy. The advantage of using carrier smoothing is apparent when comparing the velocity solutions, where accuracy is an order of magnitude better.

The next chapter is a summary of conclusions and recommendations for future study.

Chapter 9

Conclusions and Recommendations

9.1 Conclusions

For all space-related systems there is an unknown element as the real test lies in the space-flight of the design. The chance of a potential problem is minimised through thorough testing, and simulating the receiver is the best way to achieve this. The actual space-flight results of the receiver lie beyond the scope of this document.

The main conclusions from this study are:

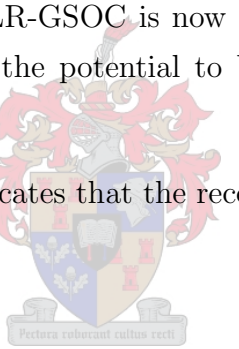
- That a LEO GPS receiver poses some challenges (high receiver dynamics, extra calculation burden), but also some simplifications (accurate position prediction, not a stand-alone device, no atmospheric delay) in operation. The unique environment of a LEO flight can be exploited to aid the operation of the receiver.
- Successful implementation of an SGPS receiver can lead to attitude determination projects using SGPS. When GPS satellites with the Civilian L2 and L5 signal come into operation, this form of attitude determination may become the method of choice for LEO satellites.
- The port of the GPS Architect software from a previous hardware implementation to a new smaller and lower power solution has been achieved. A lot of work has been done by other organisations on the Architect software for LEO use (e.g. implementing carrier phase measurements), and this software can now be used in conjunction with modern hardware.
- The space adaptation of the receiver was successful which was a world-first for the

GP4020 hardware. The idea of orbit propagation to aid signal acquisition has been proven in practise, and now also for the new receiver. Lock on multiple satellites with the SGP4 propagator can be achieved within 30 seconds or less. The ITAR restrictions have been overcome and output message adaptations have been implemented.

- The position and velocity solution of the SGPS receiver functions very well during simulation. Position accuracy of < 20 metres can be achieved if ionospheric correction is applied. For higher LEO altitudes than that tested for (445 km) the radial offset because of ionospheric error will be smaller.

- The receiver also functions with carrier-phase smoothing as developed by DLR-GSOC implemented on the modern hardware. DLR-GSOC and the ESL/US have now established a firm collaboration in the field of SGPS and as a result of the successful implementation DLR-GSOC is now migrating to the GP4020 platform. This collaborative effort has the potential to be the leader in the field of SGPS research.

- The success of the design indicates that the receiver can be used on an actual LEO mission.



9.2 Recommendations

Before the receiver is flown on a LEO mission, there are some recommendations:

- That the serial communications be refined. This includes converting the input and output sentences to binary messages. This depends on the actual hardware integration of the receiver into a satellite host. Binary messages could result in a saving of 30% in the output message sizes.

- If possible, more testing should be conducted using a GPS signal simulator. Unfortunately testing time was limited, and more varied tests should prove very useful. These should include:

    1. Testing how the receiver performs during a Cold Start (i.e. all data lost during power failure and no orbit propagation possible)
    2. Quantifying time to first fix (TTFF) values for Cold, Warm and Hot Start situations

3. Evaluation of the receiver during simulated orbit manoeuvres (Receiver antenna pointing away from earth)

- Radiation testing of the hardware both during operation for functionality, and during main power off for NVM data retention.
- A study of possible antenna designs for the LEO flight which would depend on the physical satellite host design.
- Further work on implementing the DLR-GSOC software and migrating to the new ARM Development Suite of software. DLR-GSOC is investigating the new compiler implementation and knowledge-sharing is being maintained.

Bibliography

[1] CHIARADIA, A., GILL, E., MÖNTENBRUCK, O., KUGA, H., and PRADO, A., "Algorithms for On-Board Orbit Determination using GPS OBODE-GPS." tech. rep., DLR, German Space Operations Center, Oberpfaffenhofen, Germany, 2000.

[2] GILL, E. and MÖNTENBRUCK, O., *Satellite Orbits Models, Methods, and Applications*. Springer Verlag, 2000.

[3] "2003 GPS Calendar, US National Geodetic Survey, US National Oceanic and Atmospheric Administration." http://www.ngs.noaa.gov/CORS/gpscal03.html.

[4] "GPS Interface Control Document ICD-GPS-200C." tech. rep., ARINC Research Corporation, 1993.

[5] HOOTS, F. R. and ROEHRICH, R. L., "SPACETRACK REPORT NO. 3 : Models for propagation of NORAD element sets." tech. rep., dec 1988.

[6] JOCHIM, E., GILL, E., MÖNTENBRUCK, O., and KIRSCHNER, M., "GPS based onboard and onground orbit operations for small satellites." *Acta Astronautica*, 1996, Vol. 39, No. 9, No. 9, pp. 917–922.

[7] KAPLAN, E. D. (Ed.), *Understanding GPS, Principles and Applications*. Boston, Mass: Artech House, 1996.

[8] KRUMVIEDA, K., MADHANI, P., CLOMAN, C., OLSON, E., *et al.*, "A Complete IF Software GPS Receiver: A Tutorial about the Details." tech. rep., Data Fusion Corporation, 2002.

[9] LANE, M. and HOOTS, F., "General Perturbations Theories Derived from the 1965 Lane Drag Theory, Project Space Track Report No. 2." tech. rep., Aerospace Defense Command, Peterson AFB, CO., dec 1979.

[10] LANGLEY, R., "Time, Clock and GPS." *GPS World*, November/December 1991, pp. 38–42.

[11] LEUNG, S., MÖNTENBRUCK, O., and BRUNINGA, R., "Hot Start of GPS Receivers for LEO Microsatellites." in *NAVITEC2001, ESTEC Noordwijk*, dec 2001.

[12] MITEL SEMICONDUCTOR, Cheney Manor, Swindon, Wiltshire, United Kingdom, SN2 2QW. *GPS Architect Software Design Manual*.

[13] MÖNTENBRUCK, O. and HOLT, G., "Spaceborne GPS Receiver Performance Testing." tech. rep., Space Flight Technology, German Space Operations Center (GSOC), Deutsches Zentrum fur Luft- und Raumfahrt (DLR) e.V., 2002.

[14] MÖNTENBRUCK, O., MARKGRAF, M., LEUNG, S., and GILL, E., "A GPS Receiver for Space Applications." in *ION-GPS-2001, Salt Lake City*, September 2001.

[15] PARKINSON, B. W. and JR, J. J. S. (Eds), *Global Positioning System: Theory and Applications Volume 1*. American Institute of Aeronautics and Astronautics Inc., 1996.

[16] POWELL, T. D., MARTZEN, P. D., SEDLACEK, S. B., CHAO, C.-C., SILVA, R., BROWN, A., and BELLE, G., "GPS Signals in a Geosynchronous Transfer Orbit: Falcon Gold Data Processing." tech. rep., The Aerospace Corporation, NAVSYS Corporation, United States Air Force Academy.

[17] SAKAMURA, K., *microITRON 3.0 Specification*. TRON ASSOCIATION, Katsuta Building 5F, 3-39 Mita 1-chome, Minato-ku, Tokyo 108, JAPAN, 1995.

[18] SEEBER, G., *Satellite Geodessy: Foundations, Methods and Applications*. New York, NY: Walter De Gruynter, 1993.

[19] SIGTEC NAVIGATION. *MG5000 Series GPS Receiver User Guide*.

[20] SIGTEC NAVIGATION. *multiNAV MG5021 GPS Receiver Development Kit User Guide*.

[21] SURREY SATELLITE TECHNOLOGY LIMITED. *SSTL SGR Space GPS Receiver*.

[22] "An Introduction to Thumb, Advanced RISC Machines Ltd. (ARM)." `http://www.arm.com/`.

[23] WERTZ, J. R. and LARSON, W. J. (Eds), *Space Mission Analysis and Design*. Third edition. Microcosm Press and Kluwer Academic Publishers, 1999.

[24] "WGS84: Department of Defence World Geodetic Systems 1984." tech. rep., National Imagery and Mapping Agency, January 2000.

[25] ZARLINK SEMICONDUCTOR. *GP2000 GPS Receiver Hardware Design Application Note*.

[26] ZARLINK SEMICONDUCTOR. *GP2015 GPS Receiver RF Front End*.

[27] ZARLINK SEMICONDUCTOR. *GP2021 GPS 12-Channel Correlator*.

[28] ZARLINK SEMICONDUCTOR. *GP4020 GPS Receiver Baseband Processor*.

[29] ZARLINK SEMICONDUCTOR. *GPS Orion 12 Channel GPS Receiver Design Product Brief*.

Appendix A

Coordinate Calculations

A.1 GPS Satellite Coordinate Calculations

Table A.1: *GPS Ephemeris elements*

M_0	Mean anomaly at reference time
Δn	Mean motion difference from computed value
e	Eccentricity
$A^{1/2}$	Square root of the semi-major Axis
Ω_0	Longitude of ascending node of orbit plane at weekly epoch
i_0	Inclination angle at reference time
ω	Argument of perigee
$\dot{\Omega}$	Rate of right ascension
$IDOT$	Rate of inclination angle
c_{uc}	Amplitude of the cosine harmonic correction term to the argument of latitude
c_{us}	Amplitude of the cosine harmonic correction term to the argument of latitude
c_{rc}	Amplitude of the cosine harmonic correction term to the orbit radius
c_{rs}	Amplitude of the sine harmonic correction term to the orbit radius
c_{ic}	Amplitude of the cosine harmonic correction term to the angle of inclination
c_{is}	Amplitude of the sine harmonic correction term to the angle of inclination
t_{oe}	Reference time ephemeris
$IODE$	Issue of data (ephemeris)

$\mu = 3.986005 \times 10^{14}$ metres^3/sec^2	WGS 84 value of the earth's universal gravitational parameter
$\dot{\Omega}_e = 7.2921151467 \times 10^{-5}$ rad/sec	WGS 84 value of the earth's rotation rate
$A = (\sqrt{A})^2$	Semi-major axis
$n_0 = \sqrt{\frac{\mu}{A^3}}$	Computed mean motion (rad/sec)
$t_k = t - t_{oe}$[†]	Time from ephemeris reference epoch
$n = n_0 + \Delta n$	Corrected mean motion
$M_k = M_0 + nt_k$	Mean anomaly
$M_k = E_k - e \sin E_k$	Kepler's Equation for Eccentric Anomaly (may be solved by iteration) [radians]
$\nu_k = \tan^{-1}\{\frac{\sin \nu_k}{\cos \nu_k}\}$	True anomaly
$= \tan^{-1}\{\frac{\sqrt{1-e^2}\sin E_k/(1-e\cos E_k)}{(\cos E_k - e)/(1-e\cos E_k)}\}$	
$\Phi_k = \nu_k + \omega$	Argument of latitude
$\delta u_k = c_{us} \sin 2\phi_k + c_{uc} \cos 2\phi_k$	Argument of latitude correction

Table A.2: GPS Coordinate Calculation [Page 1 of 2]

[†]t is GPS system at time of transmission, i.e. GPS time corrected for transit time (range/speed of light). Furthermore, t_k shall be the actual total time difference between the time t and the epoch time t_{oe} and must account for beginning or end of week crossovers. That is, if t_k is greater than 302400 seconds, subtract 604800 seconds from t_k. If t_k is less than -302400 seconds, add 604800 seconds to t_k.

$\delta r_k = c_{rs} \sin 2\phi_k + c_{rc} \cos 2\phi_k$	Radius correction
$\delta i_k = c_{is} \sin 2\phi_k + c_{ic} \cos 2\phi_k$	Inclination correction
$u_k = \Phi_k + \delta u_k$	Corrected argument of latitude
$r_k = A(1 - e \cos E_k) + \delta r_k$	Corrected radius
$i_k = i_0 + \delta i_k + (IDOT)t_k$	Corrected inclination
$x'_k = r_k \cos u_k$	Positions in
$y'_k = r_k \sin u_k$	orbital plane
$\Omega_k = \Omega_0 + (\dot{\Omega} - \dot{\Omega}_e)t_k - \dot{\Omega}_e t_{oe}$	Corrected longitude of ascending node
$x_k = x'_k \cos \Omega_k - y'_k \cos i_k \sin \Omega_k$	
$y_k = x'_k \sin \Omega_k + y'_k \cos i_k \cos \Omega_k$	Earth fixed coordinates
$z_k = y'_k \sin i_k$	

Table A.2: GPS Coordinate Calculation [Page 2 of 2]

Appendix B

PRN codes

B.1 PRN code phase tables

SV PRN Number	C/A-Code Tap (Chips)	C/A-Code Delay (Chips)	First 10 C/A-Chips (Octal)*
1	$2 \oplus 6$	5	1140
2	$3 \oplus 7$	6	1620
3	$4 \oplus 8$	7	1710
4	$5 \oplus 9$	8	1744
5	$1 \oplus 9$	17	1133
6	$2 \oplus 10$	18	1455
7	$1 \oplus 8$	139	1131
8	$2 \oplus 9$	140	1454
9	$3 \oplus 10$	141	1626
10	$2 \oplus 3$	251	1504
11	$3 \oplus 4$	252	1642
12	$5 \oplus 6$	254	1750
13	$6 \oplus 7$	255	1764
14	$7 \oplus 8$	256	1772
15	$8 \oplus 9$	257	1775
16	$9 \oplus 10$	258	1776
17	$1 \oplus 4$	469	1156
18	$2 \oplus 5$	470	1467
19	$3 \oplus 6$	471	1633
20	$4 \oplus 7$	472	1715

Table B.1: Code-phase assignments for the C/A-Code [Page 1 of 2]

*In the octal notation for the first 10 chips of the C/A code as shown in this column the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. For example, the first 10 chips of the SV PRN number 1 C/A-code are 1100100000.

SV PRN Number	C/A-Code Tap (Chips)	C/A-Code Delay (Chips)	First 10 C/A-Chips (Octal)*
21	$5 \oplus 8$	473	1746
22	$6 \oplus 9$	474	1763
23	$1 \oplus 3$	509	1063
24	$4 \oplus 6$	512	1706
25	$5 \oplus 7$	513	1743
26	$6 \oplus 8$	514	1761
27	$7 \oplus 9$	515	1770
28	$8 \oplus 10$	516	1774
29	$1 \oplus 6$	859	1127
30	$2 \oplus 7$	860	1453
31	$3 \oplus 8$	861	1625
32	$4 \oplus 9$	862	1712
33†	$5 \oplus 10$	863	1745
34†	$4 \oplus 10$	950 ‡	1713
35†	$1 \oplus 7$	947	1134
36†	$2 \oplus 8$	948	1456
37†	$4 \oplus 10$	950 ‡	1713

Table B.1: Code-phase assignments for the C/A-Code [Page 2 of 2]

*In the octal notation for the first 10 chips of the C/A code as shown in this column the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. For example, the first 10 chips of the SV PRN number 1 C/A-code are 1100100000.

†PRN codes 33 and 37 are reserved for other uses (e.g. ground transmitters)

‡C/A-codes 34 and 37 are identical

B.2 MATLAB Code for generating the C/A codes

```
G1 = [1 1 1 1 1 1 1 1 1 1];
G2 = [1 1 1 1 1 1 1 1 1 1];

N = 1023;
M = 8;              % number of signals
XG = zeros(M,N);
S = [[2 6];[3 7];[4 8];[5 9];[1 9];[2 10]; ...
[1 8];[2 9];[3 10];[2 3];[3 4];[5 6]];
for i = 1:N
    for k = 1:size(S,1)
        XG(k,i) = xor(xor(G2(S(k,1)),G2(S(k,2))),G1(10));
    end
    G1temp =  xor(G1(10),G1(3));
    G2temp = xor(xor(xor(G2(10),G2(9)),xor(G2(8),G2(6))), ...
      xor(G2(2),G2(3)));
    for k = 10:-1:2
        G1(k) = G1(k-1);
        G2(k) = G2(k-1);
    end
    G1(1) = G1temp;
    G2(1) = G2temp;
end
%XG contains the C/A code
```

Appendix C

Receiver Code

C.1 Coordinate Transformations

```
double GST(int GPSWeek, double GPSTOW) {
  #define jd_gps 2444244.5 // Julian Date on Jan 6, 1980 0:0:0 GMT
  double UT,TU,GMST,jd;

  // Calculate number of days since start of GPS
  double GPSDay = GPSWeek*7 + GPSTOW/(24*3600);
  jd = jd_gps+GPSDay;
  UT = Modulus(jd+0.5,1);
  jd = jd - UT;
  TU = (jd - 2451545.0)/36525;
  GMST = 24110.54841 + TU*(8640184.812866+TU*(0.093104-TU*6.2e-6));
  GMST = Modulus(GMST + 86400.0*1.00273790934*UT,86400.0);
  return 2*PI*GMST/86400.0;
}

void ECItoECEF(double omega) {
  double o_cross_r[3];
  double vi[3];

  SPVECEF.pos.x = SPVECI.pos.x*cos(omega) + SPVECI.pos.y*sin(omega);
  SPVECEF.pos.y = -SPVECI.pos.x*sin(omega) + SPVECI.pos.y*cos(omega);
  SPVECEF.pos.z = SPVECI.pos.z;
```

```
  o_cross_r[0] = -EARTH_RATE*SPVECI.pos.y;
  o_cross_r[1] = EARTH_RATE*SPVECI.pos.x;
  o_cross_r[2] = 0;

  // convert to ECEF velocity by subtracting the omega cross r term
  vi[0] = SPVECI.vel.x - o_cross_r[0];
  vi[1] = SPVECI.vel.y - o_cross_r[1];
  vi[2] = SPVECI.vel.z - o_cross_r[2];

  // now convert ECEF velocity from ECI to ECEF frame
  SPVECEF.vel.x = vi[0]*cos(omega) + vi[1]*sin(omega);
  SPVECEF.vel.y = -vi[0]*sin(omega) + vi[1]*cos(omega);
  SPVECEF.vel.z = vi[2];
}
```

C.2 Serial Communications

C.2.1 Command Task

```
void CmdTask(void)
{
    char    Buf[0x100];
    char    decodebuf[0x100],*cmdptr;
    int     msgsz;
    ER ercd;
    int i;

    SavingAlmanac = FALSE;
    LoadingAlmanac = FALSE;
    while(1)
    {
        // wait for command
        ercd = i_rcv_mbf((VP)Buf,&msgsz,cMbfRs232OutA);
        cmdptr = decodebuf;
```

```
        for (i=0;i<msgsz;i++) {
            if(Buf[i]==STX)
                cmdptr = decodebuf;
            else if(Buf[i]==ETX) {
                *cmdptr = 0x00;
                if(ValidateChecksum(&decodebuf[0])==TRUE)
                {
                    // Remove Checksum
                    *--cmdptr = 0x00;
                    *--cmdptr = 0x00;
                    SendString("CMDOK");
                    ProcessCommand(decodebuf);
                }
                else
                {
                SendString(strcat("CMDERR",decodebuf));
                cmdptr = decodebuf;
                }
            }
            else
                *cmdptr++ = Buf[i];
        }
    }
}
```

C.2.2 SendString Function

```
#define TXBUFSZ (200)

void TXBuf(char *String)
    {
    char    outputbuf[TXBUFSZ];
    ER      ercd;
    INT     msgsz;
```

```
    memcpy(&outputbuf[0],String,strlen(String));

    *String = MainPort;
    msgsz = strlen(String);
    ercd = i_tsnd_mbf(cMbfRs232In,String,strlen(String),2000);
    }

void TXString(char *String)
    {
    char buff[TXBUFSZ];
    while (strlen(String)>(TXBUFSZ-2))
        {
        memcpy(&buff[1],String,TXBUFSZ-2);
        buff[0] = '$';
        buff[TXBUFSZ-1] = 0x00;
        TXBuf(buff);
        String += TXBUFSZ-2;
        }
    memcpy(&buff[1],String,strlen(String)+1);
    TXBuf(buff);
    }

void SendString(char *String)
    {
    char buff[0x400];

    if(PC_Monitor_Mode==NMEA_MODE)
        sprintf(buff,"$%s*%2.2X\r\n",String,GetChecksum(String));
    else if(PC_Monitor_Mode==WINMON_MODE)
        sprintf(buff,"%c%s%2.2X%c",STX,String,GetChecksum(String),ETX);
    if(PC_Monitor_Mode==NMEA_MODE||PC_Monitor_Mode==WINMON_MODE)
        TXString(buff);
}
```

Appendix D

I/O specification

D.1 Input Commands

The commands are split into 3 types,

- The WINMON commands
- The LEO Commands
- The TLE up- and download commands

All commands have the same format as the output reports. The command is started with a Start-Transmission (STX) ASCII character and ended with a End-Transmission (ETX) character. The first two character is the command type, then optionally followed by the data fields and a two-digit hexadecimal checksum:

STX	c	c	x	x	..	..	x	x	cs	cs	ETX

D.1.1 WINMON Commands

AC: All Channels

The AC command sentence is used to allocate the specified satellite to all the receiver channels (see SS command). The command will only be effective whilst the software is operating in Select Satellites mode (see TM command).

Quantity	Format	Units	Range
satellite	%d	-	≥1, ≤32

CH: Active Channels

The CH command sentence sets the number of active receiver channels.

Quantity	Format	Units	Range
channels	%d	-	≥1, ≤12

CS: Cold Start

The CS command sentence has no data fields and simply initiates a software Cold Start.

DS: Deselect Satellite

The DS command sentence deselects a specified satellite and that satellite will not be used until re-selected (see RS command).

Quantity	Format	Units	Range
satellite	%d	-	≥1, ≤32

EM: Elevation Mask

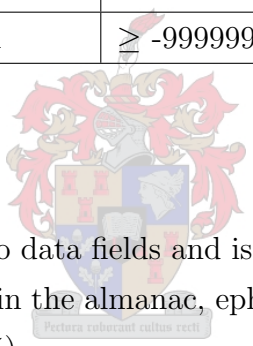
The EM command sentence sets the receiver elevation mask. Satellites below the elevation mask are excluded from the navigation solution.

Quantity	Format	Units	Range
elevation mask	%f	Degrees	≥-90.0,≤90.0

IP: Initial Position

The IP command sentence sets the receiver initial position.

Quantity	Format	Units	Range
latitude direction	%c	-	'N' or 'S'
single space	%c	-	' '
latitude degrees	%d	Degrees	≥ 0, ≤ 89
single space	%c	-	' '
latitude minutes	%f	Minutes	≥ 0, ≤ 59.9999
single space	%c	-	' '
longitude direction	%c	-	'E' or 'W'
single space	%c	-	' '
longitude degrees	%d	Degrees	≥ 0, ≤ 179
single space	%c	-	' '
longitude minutes	%f	Minutes	≥ 0, ≤ 59.9999
single space	%c	-	' '
altitude	%f	m	$\geq$ -999999999.0 and $\leq$ 999999999.0



LA: Load Almanac

The LA command sentence has no data fields and is used to inform the receiver that the next received sentences will contain the almanac, ephemeris and ionospheric/UTC model data (sentences F13, F14 and F15).

OE: Oscillator Error

The OE command sentence sets the receiver reference oscillator error in ppm.

Quantity	Format	Units	Range
oscillator error	%f	ppm	$\geq$-99.9, $\leq$99.9

PM: PDOP Mask

The PM command sentence sets the receiver PDOP mask. Satellites constellations with PDOP greater than the PDOP mask will not be used for navigation.

Quantity	Format	Units	Range
PDOP mask	%f	Degrees	$\geq$1.0, $\leq$99.9

RH: Reference Position Update

The RH command sentence has no data fields and sets the receiver reference position to the current receiver position.

RP: Reference Position

The RP command sentence sets the receiver reference position. (Only used for F00 report).

Quantity	Format	Units	Range
latitude direction	%c	-	'N' or 'S'
single space	%c	-	' '
latitude degrees	%d	Degrees	≥ 0, ≤ 89
single space	%c	-	' '
latitude minutes	%f	Minutes	≥ 0, ≤ 59.9999
single space	%c	-	' '
longitude direction	%c	-	'E' or 'W'
single space	%c	-	' '
longitude degrees	%d	Degrees	≥ 0, ≤ 179
single space	%c	-	' '
longitude minutes	%f	Minutes	≥ 0, ≤ 59.9999
single space	%c	-	' '
altitude	%f	m	$\geq$ -999999999.0 and $\leq$ 999999999.0

RS: Reselect Satellite

The RS command sentence re-selects a previously de-selected satellite (see DS command).

Quantity	Format	Units	Range
satellite	%d	-	≥ 0, ≤ 32 (0 re-selects all)

SA: Save Almanac

The SA command sentence has no data fields and is used to inform the receiver that it should transmit the sentences containing the almanac, ephemeris and ionospheric/UTC model data (sentences F13, F14 and F15).

SD: Set Date

The SD command sentence is used to set the receiver current date.

Quantity	Format	Units	Range
day	%d	-	$\geq 1, \leq 31$
single space	%c	-	' '
month	%d	-	'$\geq 1, \leq 12$
single space	%c	-	' '
year	%d	-	'$\geq 0, \leq 99$

SS: Select Satellites

The SS command sentence is used to allocate a specific satellite to a specific channel (see AC command). The command will only be effective whilst the software is operating in Select Satellites mode (see TM command).

Quantity	Format	Units	Range
satellite	%c	-	'$\geq 1, \leq 32$
single space	%c	-	' '
channel	%c	-	'$\geq 1, \leq 12$

ST: Set Time

The ST command sentence is used to set the receiver current time.

Quantity	Format	Units	Range
hour	%d	hour	$\geq 0, \leq 23$
single space	%c	-	' '
minute	%d	minute	$\geq 0, \leq 59$
single space	%c	-	' '
second	%d	second	$\geq 0, \leq 59$

TM: Track Mode

The TM command sentence is used to set the software mode for allocation of satellites to channels.

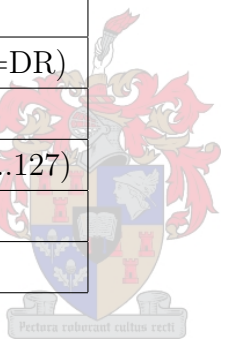
Quantity	Format	Units	Range
track mode	%d	-	Highest Elevations=1,Select Satellites=2,Cold Start=3

D.1.2 LEO Commands

DR: Data Rate [11]

Sets the output rate of the reports. Frame number indicates the sentence type (e.g. 40 for F40 message). An output rate of -1 indicates the message is off, a 0 is once-only and positive integers the message period in seconds.

Chars.	Format	Description
1	x	[STX]
2	xx	Command Id (=DR)
2	xx	Frame number
3	xxx	Output rate (-1..127)
2	xx	Checksum
1	x	[ETX]



RM: Run Mode [7]

Selects between Terrestrial and LEO mode.

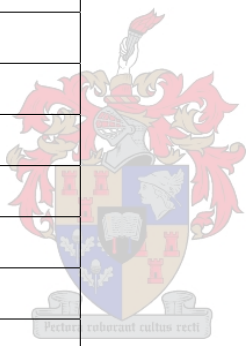
Chars.	Format	Description
1	x	[STX]
2	xx	Command Id (=RM)
1	x	Run mode (0=ground,1=ballistic,2=orbit)
2	xx	Checksum
1	x	[ETX]

D.1.3 TLE Commands

TU: TLE Upload

Command to upload the TLE values to be used in the SGP4 orbit propagator.

Quantity	Format	Units
name	%s	-
single space	%c	-
satNo	%s	-
single space	%c	-
epochYear	%d	-
single space	%c	-
epochDay	%le	-
single space	%c	-
dMM	%le	-
single space	%c	-
d2MM	%le	-
single space	%c	-
Bstar	%le	-
single space	%c	-
inclination	%le	Degrees
single space	%c	-
RAAN	%le	Degrees
single space	%c	-
ecc	%le	Degrees
single space	%c	-
argPer	%le	Degrees
single space	%c	-
MA	%le	Degrees
single space	%c	-
MM	%le	Revs per Day
single space	%c	-
revNo	%d	Revs



TD: TLE Download

The receiver should output the current TLE values.

D.2 Output Reports

D.2.1 WINMON Output Sentences

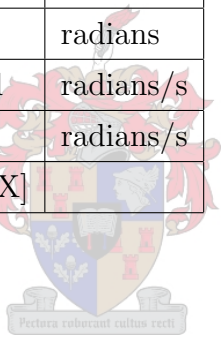
F13: Satellite Almanac Data [120 chars]

Chars	Format	Quantity	Units
1	X	[STX]	
3	XXX	Message Id (=F13)	
2	XX	PRN	-
1	X	vflg	-
3	XXX	almhlth	-
4	XXXX	refweek	weeks
6	XXXXXX	toa	sec
9	F.FFFFFFF	ecc	radians
9	±F.FFFFFF	inclin	radians/sec
15	±F.FFFFFFFFFFF	rora	$m^{1/2}$
9	FFFF.FFFF	sqrta	radians
10	±F.FFFFFFF	ratoa	radians
10	±F.FFFFFFF	argprg	radians
10	±F.FFFFFFF	manom	sec
10	±F.FFFFFFF	af0	sec/sec
15	±F.FFFFFFFFFFF	af1	
2	XX	Checksum	
1	X	[ETX]	

F14: Satellite Ephemeris Data [304 chars]

Chars	Format	Quantity	Units
1	X	[STX]	
3	XXX	Message Id (=F14)	
2	XX	PRN	-
1	X	vflg	-
6	XXXXXX	TofXmission	sec
3	XXX	s1hlth	-
1	X	codeL2	-
4	XXXX	wkn	weeks
1	X	L2Pdata	-
2	XX	ura	-
4	XXXX	iodc	-
13	±F.FFFFFFFFFF	tgd	sec
4	XXXX	tocwk	weeks
6	XXXXXX	toc	sec
13	±F.FFFFFFFFFF	af0	sec/sec
16	±F.FFFFFFFFFFFFF	af1	sec/sec^2
20	±F.FFFFFFFFFFFFFFFFF	af2	sec/sec^3
3	XXX	iode	-
8	±FFFF.FF	crs	metres
16	±F.FFFFFFFFFFFFF	deltan	radians/sec

13	±F.FFFFFFFFFF	m0	radians
12	±F.FFFFFFFFF	cuc	radians
12	F.FFFFFFFFFF	ecc	-
12	±F.FFFFFFFFF	cus	radians
11	FFFF.FFFFFF	sqrta	$\text{metres}^{1/2}$
4	XXXX	toewk	weeks
6	XXXXXX	toe	sec
1	X	fti	-
12	±F.FFFFFFFFF	cic	radians
13	±F.FFFFFFFFFF	om0	radians
12	±F.FFFFFFFFF	cis	radians
13	±F.FFFFFFFFFF	in0	radians
8	±FFFF.FF	crc	metres
13	±F.FFFFFFFFFF	olc	radians
16	±F.FFFFFFFFFFFFF	omd	radians/s
16	±F.FFFFFFFFFFFFF	idot	radians/s
1	X	[ETX]	



F15: Ionospheric/UTC Model Data [144 chars]

Chars	Format	Quantity	Units
1	X	[STX]	
3	XXX	Message Id (=F15)	
1	X	vflg	-
13	±F.FFFFFFFFFF	a0	sec
12	±F.FFFFFFFFF	a1	sec/semicircle
11	±F.FFFFFFFF	a2	sec/semicircle^2
11	±F.FFFFFFFF	a3	sec/semicircle^3
7	±XXXXXX	b0	sec
8	±XXXXXXX	b1	sec/semicircle
9	±XXXXXXXX	b2	sec/semicircle^2
9	±XXXXXXXX	b3	sec/semicircle^3
13	±F.FFFFFFFFFF	A0	sec
19	F.FFFFFFFFFFFFFFFF	A1	sec/sec
7	XXXXXXX	tot	sec
4	±XXX	dtls	sec
3	XXX	wnt	weeks
3	XXX	wnlsf	weeks
3	XXX	dn	days
4	±XXX	dtlsf	sec
1	X	[ETX]	

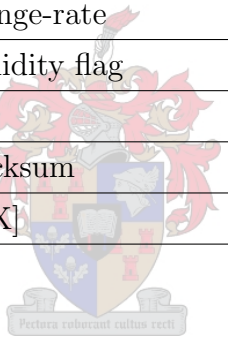
D.2.2 LEO Output Messages

F40: Navigation Data [104 chars]

Chars.	Format	Description	Units
1	X	[STX]	
3	XXX	Message Id (=F40)	
4	XXXX	GPS week	
12	FFFFFF.FFFFF	GPS seconds of week (of navigation solution)	s
2	XX	GPS-UTC	s
12	±FFFFFFFF.FF	x (WGS84)	m
12	±FFFFFFFF.FF	y (WGS84)	m
12	±FFFFFFFF.FF	z (WGS84)	m
12	±FFFFF.FFFFF	vx (WGS84)	m
12	±FFFFF.FFFFF	vy (WGS84)	m
12	±FFFFF.FFFFF	vz (WGS84)	m
1	X	Navigation status (0=no-Nav,2=3D-Nav)	
2	XX	Number of tracked satellites	
4	FF.F	PDOP	
2	XX	Checksum	
1	X	[ETX]	

F41: Pseudorange and range-rate [337 chars]

Chars.	Format	Description	Units
1	X	[STX]	
3	XXX	Message Id (=F41)	
4	XXXX	GPS week	
12	FFFFFF.FFFFF	GPS seconds of week (GPS time of observ. using current clock model)	s
2	XX	GPS-UTC	s
12	FFFFFF.FFFFF	TIC time	s
		Repeated for each of 12 channels	
2	XX	PRN	
13	FFFFFFFFFF.FF	Pseudorange	m
9	±FFFFF.FF	Range-rate	m/s
1	X	Validity flag	
2	XX	Checksum	
1	X	[ETX]	



F43: Channel Status [253 chars]

Chars.	Format	Description	Units
1	X	[STX]	
3	XXX	Message Id (=F43)	
4	XXXX	GPS week	
8	FFFFFF.F	GPS seconds of week (at message generation time)	s
2	XX	GPS-UTC	s
4	XXXX	Almanac week	
6	XXXXXX	Time of applicability	s
2	XX	PRN of last almanac frame	
1	X	Track mode (1=HighElev,2=SatSel,3=ColdStart)	
1	X	Navigation status (0=no-Nav,2=3D-Nav)	
2	XX	Number of tracked satellites	
		Repeated for each of 12 channels	
2	XX	PRN	
6	±XXXXX	Satellite Doppler (predicted)	Hz
6	±XXXXX	NCO	Hz
1	X	Subframe	
1	X	Lock indicator (c/C/B/F)	
2	XX	Signal-noise ratio (>0)	dB
2	XX	Checksum	
1	X	[ETX]	

F44: Clock Data [71 chars]

Chars.	Format	Description	Units
1	X	[STX]	
3	XXX	Message Id (=F44)	
4	XXXX	GPS week	
12	FFFFFF.FFFFF	GPS seconds of week (at message generation time)	s
2	XX	GPS-UTC	s
8	XXXXXXXX	TIC count	
8	XXXXXXXX	Real Time Clock count	s
4	XXXX	Extrapolated boot time (GPS week)	
12	FFFFFF.FFFFF	Extrapolated boot time (GPS seconds of week) [s]	s
6	±F.FFF	Clock drift	μs/s
8	XXXXXXXX	Time of applicability of clock model (TIC)	
2	XX	Checksum	
1	X	[ETX]	

F48: Configuration and Status Parameters [48 chars]

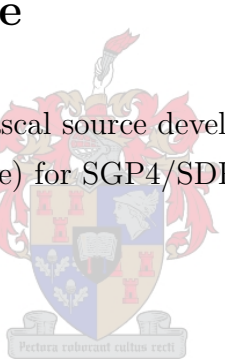
Chars.	Format	Description	Units
1	X	[STX]	
3	XXX	Message Id (=F48)	
4	XXXX	GPS week	
8	FFFFFF.F	GPS seconds of week (at output)	s
2	XX	GPS-UTC	s
4	XXXX	Almanac week	
5	±XXXX	Doppler offset [Hz]	Hz
1	X	Mode (0=default,1=rocket,2=orbit)	
1	X	Output format (0=default,1=extended)	
1	X	Update rate of navigation and display task	Hz
2	XX	Spare CPU capacity	%
3	±XX	Elevation mask	deg
2	XX	PDOP mask	
8	FFFFFF.F	Launch time (GPS seconds of week)	s
1	X	[ETX]	

Appendix E

Java Source Code

E.1 SGP4 Java Code

This code was adapted from the Pascal source developed by Dr TS Kelso (http://www.celestrak.com/software) for SGP4/SDP4 satellite tracking.




```
import java.io.*;
import java.util.*;
class SGP4Conv {

  static private final double ae       = 1;
  static private final double tothrd   = (double)2/3;
  // {Earth equatorial radius - kilometers (WGS '72)}
  static private final double xkmper   = 6378.135;
  // {Earth flattening (WGS '72)}
  static private final double f        = 1/298.26;
  // {Earth flattening (WGS '72)}
  static private final double ge       = 398600.8;
  // {J2 harmonic (WGS '72)}
  static private final double J2       = 1.0826158E-3;
  // {J3 harmonic (WGS '72)}
  static private final double J3       = -2.53881E-6;
  // {J4 harmonic (WGS '72)}
  static private final double J4       = -1.65597E-6;
  static private final double ck2      = J2/2;
```

```
static private final double ck4      = -3*J4/8;
static private final double xj3      = J3;
static private final double qo       = ae + 120/xkmper;
static private final double s        = ae + 78/xkmper;
static private final double e6a      = 1E-6;
// {Minutes per day}
static private final double xmnpda   =  1440.0;
// {Sqrt(ge) ER^3/min^2}
static private final double xke      = Math.sqrt(3600*ge/Cube(xkmper));
// {(qo-s)^4 ER^4}
static private final double qoms2t   = Sqr(Sqr(qo-s));
static private final double OmegaeDot = 7.2921151467e-5;

private static double cos(double a) {
  return Math.cos(a);
}

private static double sin(double a) {
  return Math.sin(a);
}

private static double pow(double a,double b) {
  return Math.pow(a,b);
}

private static double sqrt(double a) {
  return Math.sqrt(a);
}

private static double floor(double a) {
  return Math.floor(a);
}

private static double Radians(double deg) {
  return deg/180*Math.PI;
}

private static double Sqr(double a) {
  return a*a;
```

```
}

private static double Cube(double a) {
  return a*a*a;
}

private static int Trunc(double a) {
return (int)Math.floor(a);
}

private static double Modulus(double arg1,double arg2) {
  double modu;
  modu = arg1 - Trunc(arg1/arg2) * arg2;
  if (modu >= 0)
    return modu;
  else
    return modu + arg2;
}

private static double Julian_Date_of_Epoch(double epYear,double epDay) {
  double year,day;

  // { Modification to support Y2K }
  // { Valid 1957 through 2056 }
  year = epYear;
  if (year < 57)
    year = year + 2000;
  else
    year = year + 1900;
  // { End modification }
  day  = epDay;
  return Julian_Date_of_Year(year) + day;
}

private static double Julian_Date_of_Year(double year) {
  // { Astronomical Formulae for Calculators, Jean Meeus, pages 23-25 }
  // { Calculate Julian Date of 0.0 Jan year }
  int A,B;
```

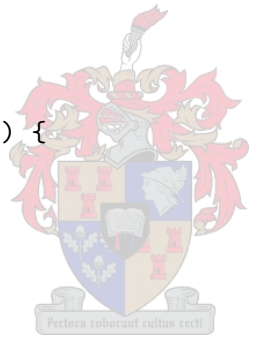
```
  year = year - 1;
  A = Trunc(year/100);
  B = 2 - A + Trunc(A/4);
  return Trunc(365.25 * year) + Trunc(30.6001 * 14) + 1720994.5 + B;
}

static double xndt2o,xndd6o,bstar,xincl,xnodeo,eo,omegao,xmo,xno,epoch;
static double julian_epoch;

private static void Convert_Satellite_Data(Keplerian kep) {

  double a1,ao,del1,delo,xnodp,temp;
  final double twopi = Math.PI*2;

  epoch    = kep.fullEpoch;
  julian_epoch = Julian_Date_of_Epoch(kep.epochYear,kep.epoch);
  xndt2o   = kep.dMM;
  xndd6o   = kep.d2MM;
  bstar    = kep.Bstar;
  xincl    = kep.inclination;
  xnodeo   = kep.RAAN;
  eo       = kep.ecc;
  omegao   = kep.argPer;
  xmo      = kep.MA;
  xno      = kep.MM;

  //{* Convert to proper units *}
  bstar    = bstar/ae;
  xnodeo   = Radians(xnodeo);
  omegao   = Radians(omegao);
  xmo      = Radians(xmo);
  xincl    = Radians(xincl);
  xno      = xno*twopi/xmnpda;
  xndt2o   = xndt2o*twopi/Sqr(xmnpda);
  xndd6o   = xndd6o*twopi/Cube(xmnpda);

  // {* Determine whether Deep-Space Model is needed *}
  a1 = pow(xke/xno,tothrd);
  temp = 1.5*ck2*(3*Sqr(cos(xincl))-1)/pow(1 - eo*eo,1.5);
```

```
    del1 = temp/(a1*a1);
    ao = a1*(1 - del1*(0.5*tothrd + del1*(1 + 134/81*del1)));
    delo = temp/(ao*ao);
    xnodp = xno/(1 + delo);

  }

  public static double[] SGP4(Keplerian kep, double tsince) {

    int iflag;

    double a1,a3ovk2,ao,aodp,aycof,betao,betao2,c1,c1sq,c2,c3,c4;
    double c5,coef,coef1,cosio,d2,d3,d4,del1,delmo,delo,eeta,eosq;
    double eta,etasq,isimp,omgcof,omgdot,perige,pinvsq,psisq,qoms24;
    double s4,sinio,sinmo,t2cof,t3cof,t4cof,t5cof,temp,temp1,temp2,temp3,theta2;
    double theta4,tsi,x1m5th,x1mth2,x3thm1,x7thm1,xhdot1,xlcof,xmcof;
    double xmdot,xnodcf,xnodot,xnodp;

    double cosuk,sinuk,rfdotk,vx,vy,vz,ux,uy,uz,xmy,xmx,
    cosnok,sinnok,cosik,sinik,rdotk,xinck,xnodek,uk,rk,
    cos2u,sin2u,u,sinu,cosu,betal,rfdot,rdot,r,pl,elsq,
    esine,ecose,epw,temp6,temp5,temp4,cosepw,sinepw,
    capu,ayn,xlt,aynl,xll,axn,xn,beta,xl,e,a,tfour,
    tcube,delm,delomg,templ,tempe,tempa,xnode,tsq,xmp,
    omega,xnoddf,omgadf,xmdf,x,y,z,xdot,ydot,zdot;

    // Recover original mean motion (xnodp) and semimajor axis (aodp)
    // from input elements.
    a1 = pow(xke/xno,tothrd);
    cosio = cos(xincl);
    theta2 = cosio*cosio;
    x3thm1 = 3*theta2 - 1;
    eosq = eo*eo;
    betao2 = 1 - eosq;
    betao = sqrt(betao2);
    del1 = 1.5*ck2*x3thm1/(a1*a1*betao*betao2);
    ao = a1*(1 - del1*(0.5*tothrd + del1*(1 + 134/81*del1)));
    delo = 1.5*ck2*x3thm1/(ao*ao*betao*betao2);
    xnodp = xno/(1 + delo);
```

```
aodp = ao/(1 - delo);

//{ Initialization }
// { For perigee less than 220 kilometers, the isimp flag is set and
// the equations are truncated to linear variation in sqrt a and
// quadratic variation in mean anomaly.  Also, the c3 term, the
// delta omega term, and the delta m term are dropped. }

isimp = 0;
if ((aodp*(1 - eo)/ae) < (220/xkmper + ae))
  isimp = 1;
// { For perigee below 156 km, the values of s and qoms2t are altered. }
s4 = s;
qoms24 = qoms2t;
perige = (aodp*(1 - eo) - ae)*xkmper;

pinvsq = 1/(aodp*aodp*betao2*betao2);
tsi = 1/(aodp - s4);
eta = aodp*eo*tsi;
etasq = eta*eta;
eeta = eo*eta;
psisq = Math.abs(1 - etasq);
coef = qoms24*pow(tsi,4);
coef1 = coef/pow(psisq,3.5);
c2 = coef1*xnodp*(aodp*(1 + 1.5*etasq + eeta*(4 + etasq))
  + 0.75*ck2*tsi/psisq*x3thm1*(8 + 3*etasq*(8 + etasq)));
c1 = bstar*c2;
sinio = sin(xincl);
a3ovk2 = -xj3/ck2*pow(ae,3);
c3 = coef*tsi*a3ovk2*xnodp*ae*sinio/eo;
x1mth2 = 1 - theta2;
c4 = 2*xnodp*coef1*aodp*betao2*(eta*(2 + 0.5*etasq)
  + eo*(0.5 + 2*etasq) - 2*ck2*tsi/(aodp*psisq)
  *(-3*x3thm1*(1 - 2*eeta + etasq*(1.5 - 0.5*eeta))
  + 0.75*x1mth2*(2*etasq - eeta*(1 + etasq))*cos(2*omegao)));
c5 = 2*coef1*aodp*betao2*(1 + 2.75*(etasq + eeta) + eeta*etasq);
theta4 = theta2*theta2;
temp1 = 3*ck2*pinvsq*xnodp;
temp2 = temp1*ck2*pinvsq;
```

```
temp3 = 1.25*ck4*pinvsq*pinvsq*xnodp;
xmdot = xnodp + 0.5*temp1*betao*x3thm1
  + 0.0625*temp2*betao*(13 -78*theta2 + 137*theta4);
x1m5th = 1 - 5*theta2;
omgdot = -0.5*temp1*x1m5th + 0.0625*temp2*(7 - 114*theta2 +395*theta4)
  + temp3*(3 - 36*theta2 + 49*theta4);
xhdot1 = -temp1*cosio;
xnodot = xhdot1 + (0.5*temp2*(4 - 19*theta2)
 + 2*temp3*(3 - 7*theta2))*cosio;
omgcof = bstar*c3*cos(omegao);
xmcof = -tothrd*coef*bstar*ae/eeta;
xnodcf = 3.5*betao2*xhdot1*c1;
t2cof = 1.5*c1;
xlcof = 0.125*a3ovk2*sinio*(3 + 5*cosio)/(1 + cosio);
aycof = 0.25*a3ovk2*sinio;
delmo = pow(1 + eta*cos(xmo),3);
sinmo = sin(xmo);
x7thm1 = 7*theta2 - 1;

iflag = 0;

// { Update for secular gravity and atmospheric drag. }

xmdf = xmo + xmdot*tsince;
omgadf = omegao + omgdot*tsince;
xnoddf = xnodeo + xnodot*tsince;
omega = omgadf;
xmp = xmdf;
tsq = tsince*tsince;
xnode = xnoddf + xnodcf*tsq;
tempa = 1 - c1*tsince;
tempe = bstar*c4*tsince;
templ = t2cof*tsq;

a = aodp*Sqr(tempa);
e = eo - tempe;
xl = xmp + omega + xnode + xnodp*templ;
beta = sqrt(1 - e*e);
xn = xke/pow(a,1.5);
```

```
// { Long period periodics }
axn = e*cos(omega);
temp = 1/(a*beta*beta);
xll = temp*xlcof*axn;
aynl = temp*aycof;
xlt = xl + xll;
ayn = e*sin(omega) + aynl;

// { Solve Kepler's Equation }
capu = Modulus(xlt - xnode,2*Math.PI);
temp2 = capu;
temp4 = 0;
temp5 = 0;
temp6 = 0;
sinepw = 0;
cosepw = 0;
for(int k = 0; k < 10; k++) {
  sinepw = sin(temp2);
  cosepw = cos(temp2);
  temp3 = axn*sinepw;
  temp4 = ayn*cosepw;
  temp5 = axn*cosepw;
  temp6 = ayn*sinepw;
  epw = (capu - temp4 + temp3 - temp2)/(1 - temp5 - temp6) + temp2;
  if (Math.abs(epw - temp2) <= e6a)
    break;
  else
    temp2 = epw;
}

// { Short period preliminary quantities }
ecose = temp5 + temp6;
esine = temp3 - temp4;
elsq = axn*axn + ayn*ayn;
temp = 1 - elsq;
pl = a*temp;
r = a*(1 - ecose);
temp1 = 1/r;
```



```
rdot = xke*sqrt(a)*esine*temp1;
rfdot = xke*sqrt(pl)*temp1;
temp2 = a*temp1;
betal = sqrt(temp);
temp3 = 1/(1 + betal);
cosu = temp2*(cosepw - axn + ayn*esine*temp3);
sinu = temp2*(sinepw - ayn - axn*esine*temp3);
u = Math.atan2(sinu,cosu);
sin2u = 2*sinu*cosu;
cos2u = 2*cosu*cosu - 1;
temp = 1/pl;
temp1 = ck2*temp;
temp2 = temp1*temp;

//{ Update for short periodics }
rk = r*(1 - 1.5*temp2*betal*x3thm1) + 0.5*temp1*x1mth2*cos2u;
uk = u - 0.25*temp2*x7thm1*sin2u;
xnodek = xnode + 1.5*temp2*cosio*sin2u;
xinck = xincl + 1.5*temp2*cosio*sinio*cos2u;
rdotk = rdot - xn*temp1*x1mth2*sin2u;
rfdotk = rfdot + xn*temp1*(x1mth2*cos2u + 1.5*x3thm1);

// { Orientation vectors }
sinuk = sin(uk);
cosuk = cos(uk);
sinik = sin(xinck);
cosik = cos(xinck);
sinnok = sin(xnodek);
cosnok = cos(xnodek);
xmx = -sinnok*cosik;
xmy = cosnok*cosik;
ux = xmx*sinuk + cosnok*cosuk;
uy = xmy*sinuk + sinnok*cosuk;
uz = sinik*sinuk;
vx = xmx*cosuk - cosnok*sinuk;
vy = xmy*cosuk - sinnok*sinuk;
vz = sinik*cosuk;

// { Position and velocity }
```

```
    double[] pos = new double[3];
    double[] vel = new double[3];
    x = rk*ux;  pos[0] = x;
    y = rk*uy;  pos[1] = y;
    z = rk*uz;  pos[2] = z;
    xdot = rdotk*ux + rfdotk*vx;  vel[0] = xdot;
    ydot = rdotk*uy + rfdotk*vy;  vel[1] = ydot;
    zdot = rdotk*uz + rfdotk*vz;  vel[2] = zdot;

    for(int i=0;i <= 2;i++) {
      pos[i] = pos[i]*xkmper;    // {kilometers}
      vel[i] = vel[i]*xkmper/60; // {kilometers/second}
    }

    double[] rettemp = {pos[0],pos[1],pos[2],vel[0],vel[1],vel[2]};
    return rettemp;

  }

  public static double[] posLLA(double[] posXYZ) {

    double X = posXYZ[0];
    double Y = posXYZ[1];
    double Z = posXYZ[2];

    double a = 6378.1370;  // km             {WGS-84 earth's semi major axis}
    double b = 6356.7523142;// km             {WGS-84 earth's semi minor axis}

    double p = Math.sqrt(X*X + Y*Y);
    double theta = Math.atan(Z*a/(p*b));

    double e2 = (a*a-b*b)/(a*a);
    double e_2 = (a*a - b*b)/(b*b);

    double phi = Math.atan((Z+e_2*b*Math.pow(Math.sin(theta),3))/
      (p-e2*a*Math.pow(Math.cos(theta),3)));
    double lambda = Math.atan2(Y,X);

    double N = a/Math.sqrt(1-e2*Math.pow(Math.sin(phi),2));
```

```
    double h = p/Math.cos(phi)-N;

    double[] temp = {lambda,phi,h};
    return temp;

  }

  public static void main(String[] args) {

    try {
      Keplerian kep;
      kep = ReadKep.readOne("tubsat.tle");
      Convert_Satellite_Data(kep);
      kep.display();
      BufferedWriter out =
      new BufferedWriter(new FileWriter("LEO_ECI.dat"));

      TimeZone utc = TimeZone.getTimeZone("UT");

      // Epoch of Keplerian Elements
      Calendar epoch = Calendar.getInstance(utc);
      epoch.set(Calendar.DAY_OF_YEAR,(int)Math.floor(kep.epoch));
      if(kep.epochYear < 70)
        epoch.set(Calendar.YEAR,2000+(int)Math.floor(kep.epochYear));
      else
        epoch.set(Calendar.YEAR,1900+(int)Math.floor(kep.epochYear));
      epoch.set(Calendar.HOUR_OF_DAY,(int)(kep.epoch*24 % 24));
      epoch.set(Calendar.MINUTE,(int)(kep.epoch*24*60 % 60));
      epoch.set(Calendar.SECOND,(int)(kep.epoch*24*60*60 % 60));
      epoch.set(Calendar.MILLISECOND,(int)(kep.epoch*24*60*60*1000 % 1000));

      System.out.print("Epoch Date/Time: ");
      System.out.print(epoch.get(Calendar.YEAR)+"-");
      System.out.print(epoch.get(Calendar.MONTH)+1+"-");
      System.out.print(epoch.get(Calendar.DAY_OF_MONTH)+" ");
      System.out.print(epoch.get(Calendar.HOUR_OF_DAY)+":");
      System.out.print(epoch.get(Calendar.MINUTE)+":");
      System.out.print(epoch.get(Calendar.SECOND)+"."
       +epoch.get(Calendar.MILLISECOND)+" ");
```

```
System.out.println(epoch.getTimeZone().getDisplayName());

Calendar now = Calendar.getInstance(utc);
now.set(now.MILLISECOND,0);

System.out.print("Current Date/Time: ");
System.out.print(now.get(Calendar.YEAR)+"-");
System.out.print(now.get(Calendar.MONTH)+1+"-");
System.out.print(now.get(Calendar.DAY_OF_MONTH)+" ");
System.out.print(now.get(Calendar.HOUR_OF_DAY)+":");
System.out.print(now.get(Calendar.MINUTE)+":");
System.out.print(now.get(Calendar.SECOND)+"."
 +now.get(Calendar.MILLISECOND)+" ");
System.out.println(now.getTimeZone().getDisplayName());

final int maxk = 3000; // number of simulated coordinates
final double dt = 30;  // time between each calculation in seconds

char[] cbuf = (maxk+" "+dt).toCharArray();
out.write(cbuf,0,cbuf.length);
out.newLine();

for(int k=0; k < maxk; k++) {
  long simtime = (long)(now.getTime().getTime()+k*1000*dt); // msec

  Calendar simCal = Calendar.getInstance(utc);
  Date simDate = new Date(simtime);
  simCal.setTime(simDate);

  double tsince = ((double)(simtime-epoch.getTime().getTime()))/1000/60;
  double[] pv = SGP4(kep,tsince);

  int y = simCal.get(Calendar.YEAR);
  int m = simCal.get(Calendar.MONTH)+1;
  int d = simCal.get(Calendar.DAY_OF_MONTH);
  int h = simCal.get(Calendar.HOUR_OF_DAY);
  int mins = simCal.get(Calendar.MINUTE);
  double seconds = simCal.get(Calendar.SECOND)
    +(double)simCal.get(Calendar.MILLISECOND)/1000;
```

```
        cbuf = (y+"-"+m+"-"+d+" "+h+":"+mins+":"+seconds).toCharArray();
        out.write(cbuf,0,cbuf.length);
        cbuf = (" "+pv[0]+" "+pv[1]+" "+pv[2]+" "
          +pv[3]+" "+pv[4]+" "+pv[5]).toCharArray();
        out.write(cbuf,0,cbuf.length);
        out.newLine();

      }
      out.close();
    } catch(IOException E) {
      System.out.println("File not found or file error");
      E.printStackTrace();
    }
  }
}
```

Appendix F

Simulator Results

F.1 Pseudorange Measurements

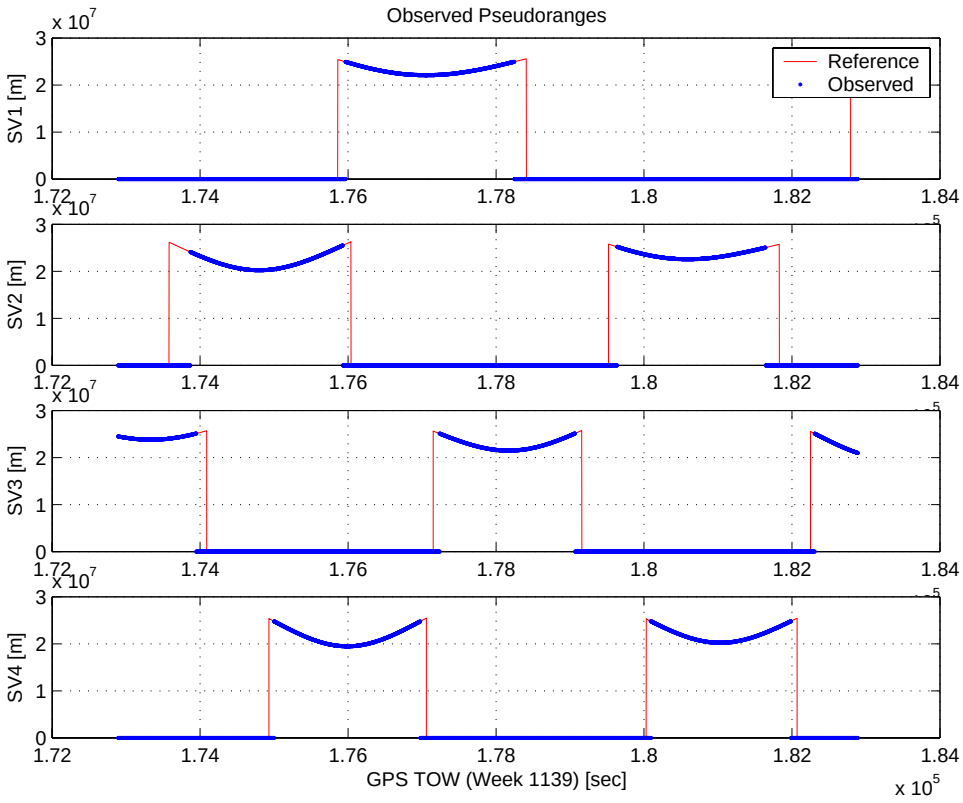


Figure F.1: *Pseudorange Measurement SV 1 to 4*

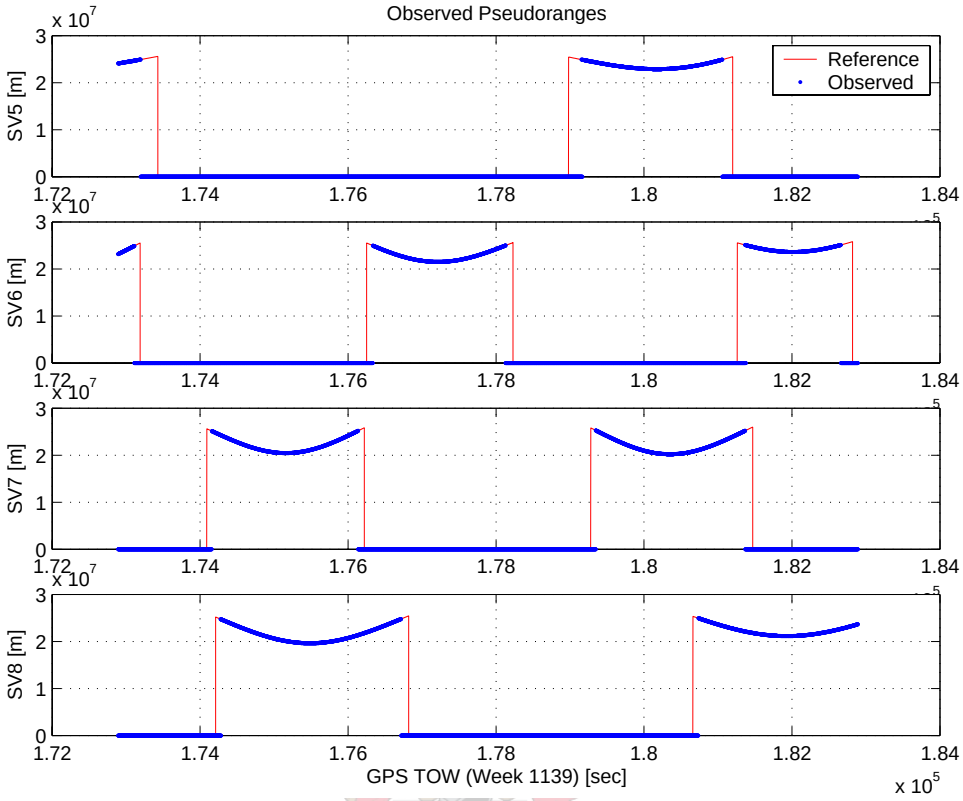


Figure F.2: *Pseudorange Measurement V 5 to 8*

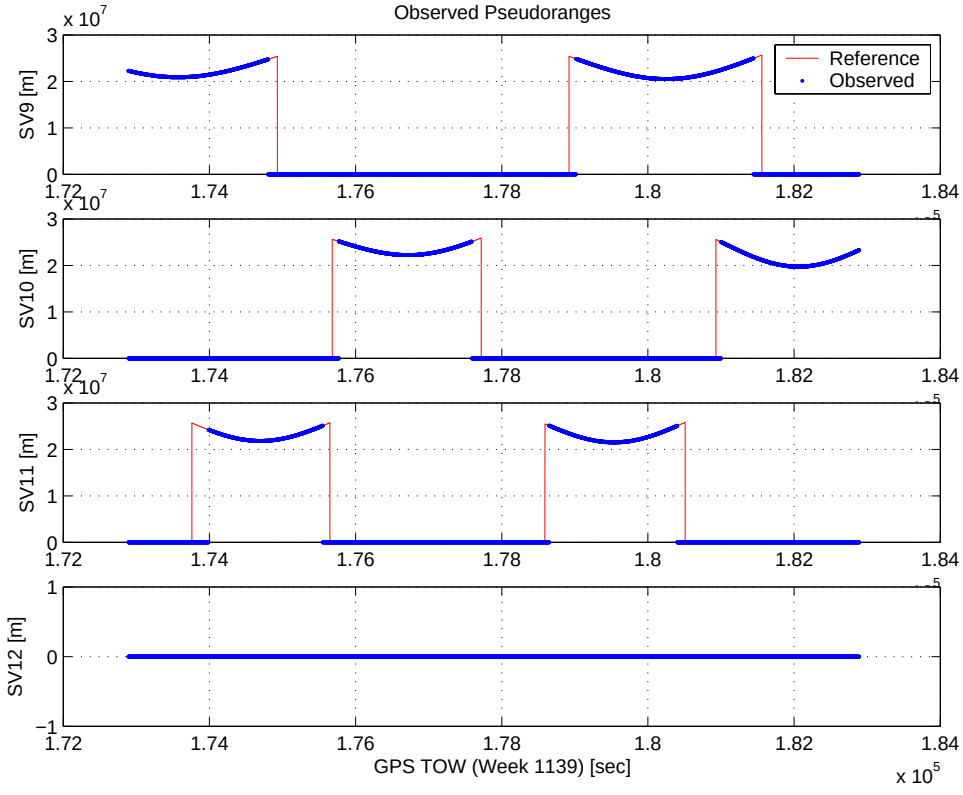


Figure F.3: *Pseudorange Measurement SV 9 to 12*

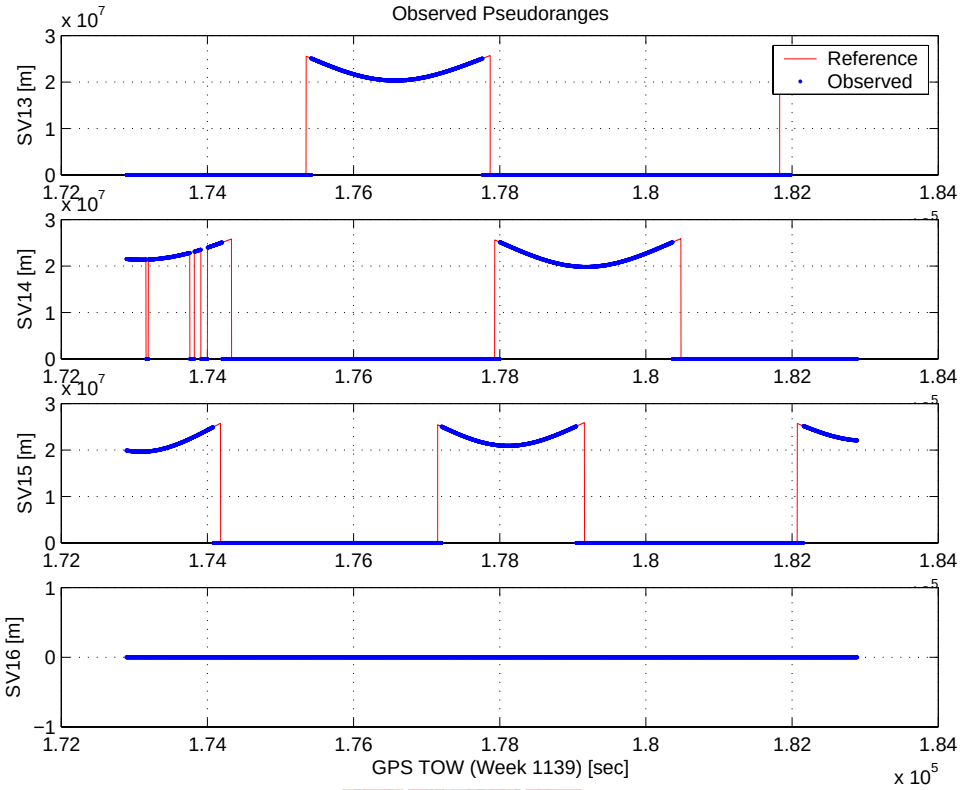


Figure F.4: *Pseudorange Measurement SV 13 to 16*

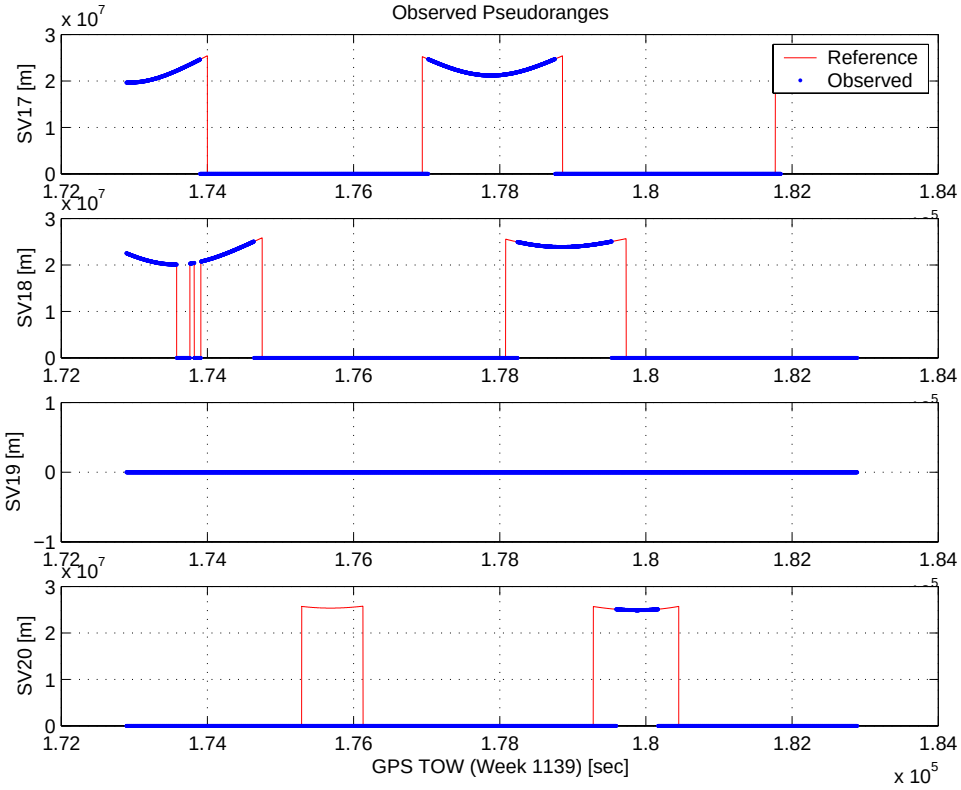


Figure F.5: *Pseudorange Measurement SV 17 to 20*

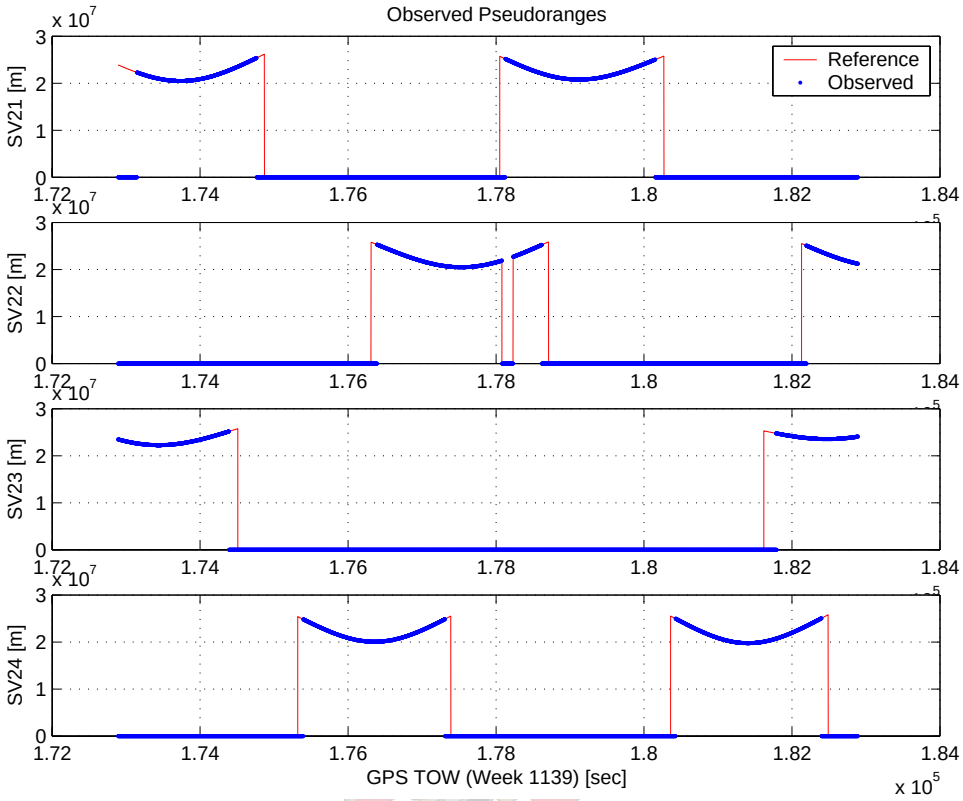


Figure F.6: *Pseudorange Measurement SV 21 to 24*

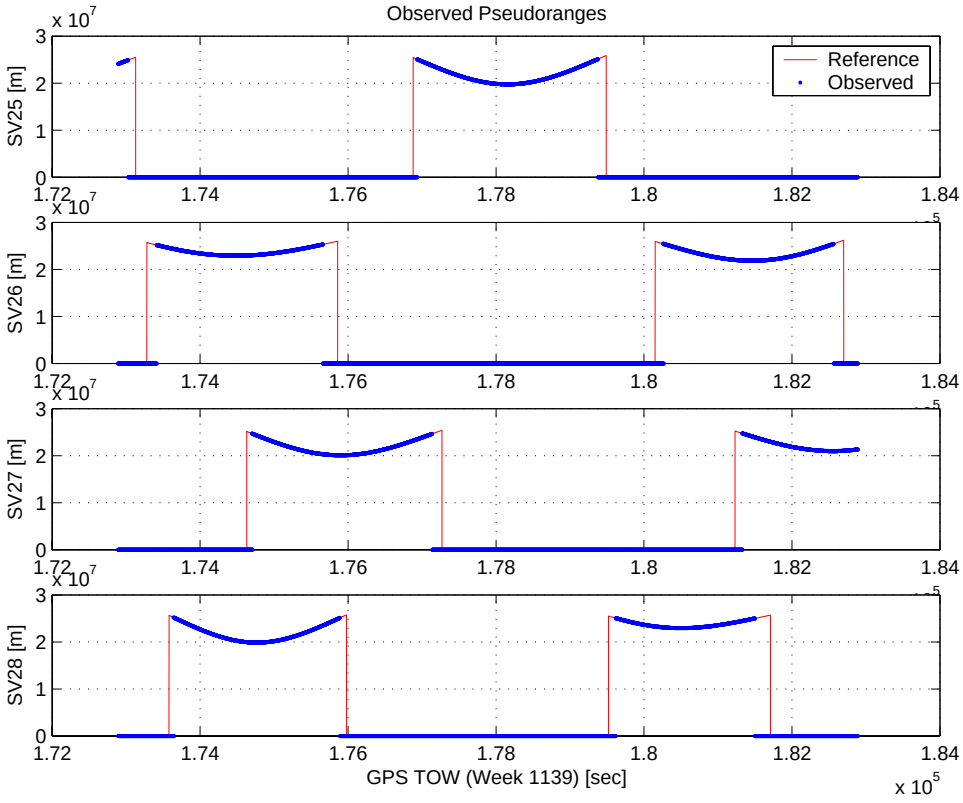


Figure F.7: *Pseudorange Measurement SV 25 to 28*

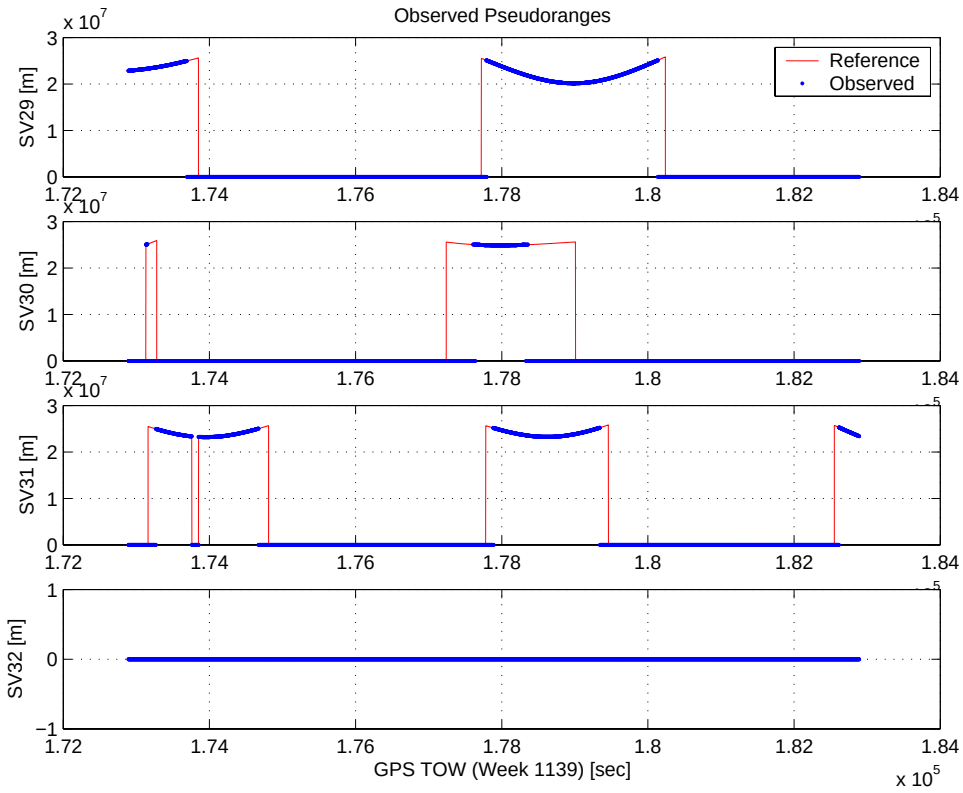


Figure F.8: *Pseudorange Measurement SV 29 to 32*

Appendix G

Receiver Model Code

G.1 MATLAB Code


```
% GPS sample signal generator
% Last Revision: 20/11/2002
% Author: BJ Nortier
% bnortier@sun.ac.za

clc;
close all;
clear all;

% Generate the PRN:
G1 = int8([1 1 1 1 1 1 1 1 1 1]);
G2 = int8([1 1 1 1 1 1 1 1 1 1]);

N = 1023;
M = 8;              % number of signals

XG = uint8(zeros(M,N));
S = uint8([[2 6];[3 7];[4 8];[5 9];[1 9];[2 10];[1 8];...
        [2 9];[3 10];[2 3];[3 4];[5 6]]);

for i = 1:N
    for k = 1:size(S,1)
```

```
        XG(k,i) = xor(xor(G2(S(k,1)),G2(S(k,2))),G1(10));
    end
    G1temp = uint8(xor(G1(10),G1(3)));
    G2temp = uint8(xor(xor(xor(G2(10),G2(9)),xor(G2(8),G2(6))), ...
        xor(G2(2),G2(3))));
    for k = 10:-1:2
        G1(k) = G1(k-1);
        G2(k) = G2(k-1);
    end
    G1(1) = G1temp;
    G2(1) = G2temp;
end

display forend

L1 = 1575.42e6;
IF = 1.5e6;
T = 1/double(IF);           % Carrier period

fcodeCA = 1.023e6   % [MHz] - CA chip frequency
TcodeCA = 1/fcodeCA;% [s] - CA chip period
clear fcodeCA;

fcodeP = 10.23e6;   % [MHz] - P chip frequency
TcodeP = 1/fcodeP;  % [s] - P chip period
clear fcodeP;

codelengthCA = 1023;

codephaseCAAll = floor(1023*rand(12,1))
codeCA = int8(1-2*double(XG)); clear XG;

phase = (2*pi*rand(M,1)-pi);

codelengthP = 10000;
codeP = int8(1-2*round(rand(12,codelengthP)));

L = 10; % data length
```

```
dataperiod = double(TcodeCA)*double(codelengthCA)*20; % [s]
sampletime = double(T(1))/4 ;
t = single(0:sampletime:((dataperiod*L)-sampletime));

data = int8(2*round(rand(M,L))-1);

tonecodeCA = single(0:sampletime:(TcodeCA*codelengthCA));
tonecodeCA = tonecodeCA(1:(size(tonecodeCA,2)-1));

for k = 1:M
    codetimeCAAll(k,:) = codeCA(k,mod(floor(mod(double(t)/...
        (TcodeCA*codelengthCA)...
        *codelengthCA,codelengthCA))+codephaseCAAll(k),codelengthCA)+1);
    codetimePAll(k,:) = codeP(k,floor(mod(double(t)/(TcodeP*codelengthP)...
        *codelengthP,codelengthP))+1);
end

clear codeP;

datatime = single(data(:,floor((double(t)/dataperiod)+1)));

N = size(t,2);
xt = single(double(datatime).*double(codetimeCAAll(1:M,:)).*sin(...
    2*pi*IF*ones(M,1)*double(t) + phase*ones(1,N))+1/sqrt(2)*double(...
    codetimePAll(1:M,:)).*cos(...
    2*pi*IF*ones(M,1)*double(t) + phase*ones(1,N)));

clear codetimePAll,codetimeCAAll;

Xt = sum(xt,1);
noise = randn(1,N);
Xt = Xt+noise;

clear xt;
clear datatime;

integratetime = .5e-4;
itime = TcodeCA*1023;
```

```
rec = [double(t)' Xt'];

codeCA = double(codeCA);
clear codetimeCAAll
clear codetimePAll
clear t
clear noise
clear code
clear Xt
clear tonecodeCA
clear codeP

if abs(min(rec(:,2))) > max(rec(:,2))
    rec(:,2) = rec(:,2)/(abs(min(rec(:,2))));
else
    rec(:,2) = rec(:,2)/(max(rec(:,2)));
end
level = 0.3;
rectemp = rec;
rec(:,2) = ...
    (rec(:,2)<-level)*-3 + ...
    ((rec(:,2)>=-level) & (rec(:,2)<0))*-1 + ...
    ((rec(:,2)<=level) & (rec(:,2)>=0))*1 + ...
    (rec(:,2)>level)*3;
rec(:,2) == rec(:,2)/3;

noelements = size(rec,1);
if noelements > 2000000
    noelements = 2000000;
end
f = fopen('..\data\samplesetup.txt','w+');
fprintf(f,'%d ',noelements);
fprintf(f,'%e ',sampletime);
fprintf(f,'%e ',integratetime);
fprintf(f,'%e ',itime);
fprintf(f,'%d ',double(codephaseCAAll(1)));
fprintf(f,'%e ',IF);
fclose(f);
```

```
f = fopen('..\data\sampledata.bin','wb+');
fwrite(f,rec(:,2),'int8');
fclose(f);
```

G.2 Simulink Model

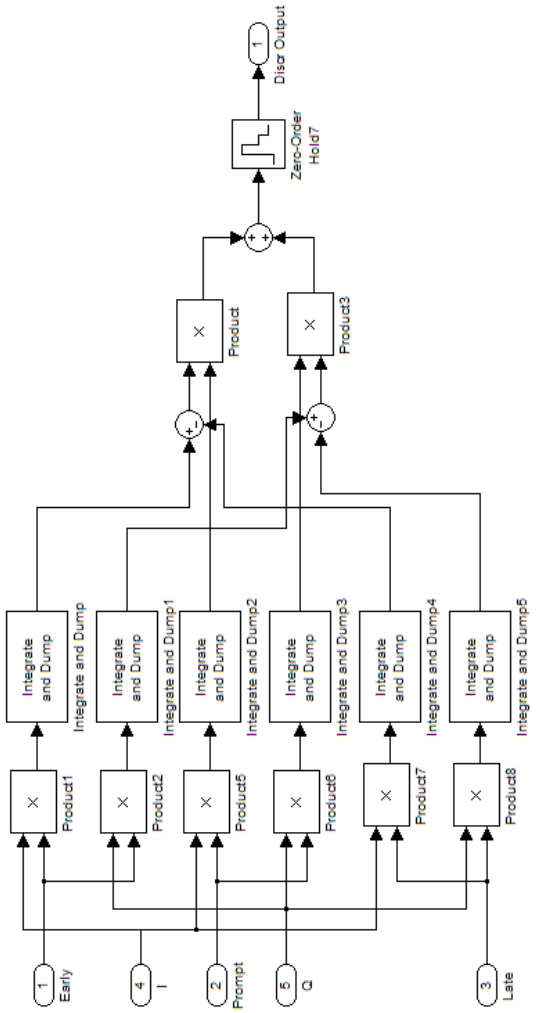


Figure G.1: *Simulink CorrelateAndHold Subsystem*

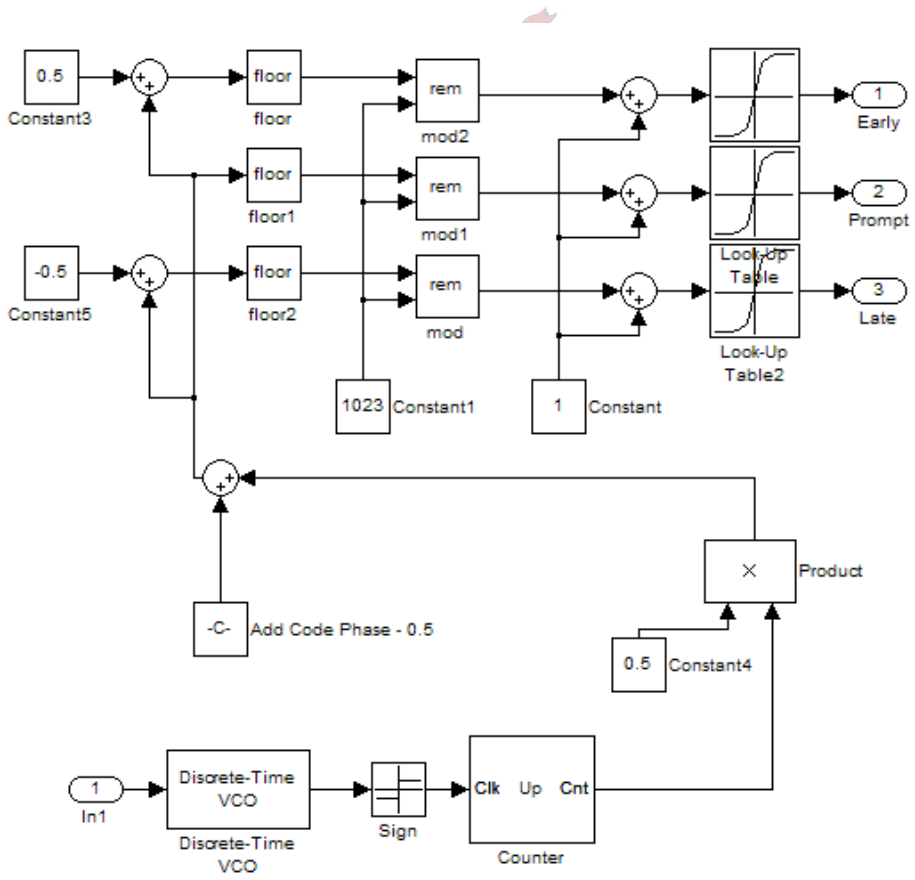


Figure G.2: *Simulink CodeGenrator Subsystem*

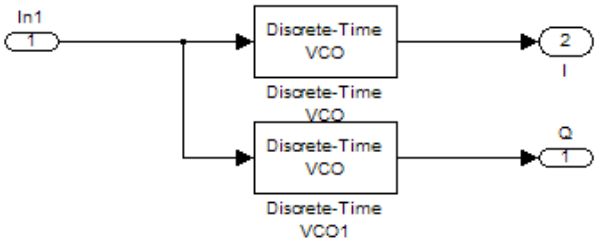


Figure G.3: *Simulink SinAndCos VCO Subsystem*