



THE UNIVERSITY OF
NEW SOUTH WALES

Taste of Summer Research Scholarships, 2006/7

Open Source Software Development for UNSW FPGA-based GPS Receiver

School of Surveying and Spatial Information Systems



<i>Supervisor</i>	<i>Peter Mumford</i>
<i>Student Name</i>	<i>Yong Heo</i>
<i>Student ID</i>	<i>3088486</i>
<i>Contact</i>	<u><i>yongh@cse.unsw.edu.au</i></u>

Acknowledgements

“You did not choose me, but I chose you and appointed you to go and bear fruit – fruit that will last.”

- *John 15:16*

I would like to give thanks to Lord Jesus for His loving kindness, grace, mercy and provision. I also thank Him for the strength He has given me.

Also, I would like to acknowledge my appreciation to a number of people who have been supporting me in many different ways throughout this project period:

Peter Mumford, my project supervisor: I would like to express my great appreciation to him who has been keeping me on track throughout this project and providing his assistance in providing me with his valuable time and input

Sun Hwa, Choi, my lovely girl-friend: I would like to special thanks her for her patience and her support.

Rish, my colleague from RMIT university: I would like to thank him for his helping me during this project with valuable feedback for a report and a poster.

Kwang, Heo: my sweet brother: I would like to especially express my gratitude for his encouragement and feedback.

Finally, I sincerely thanks to my father, Young-Soo, Heo and my mother, Kyung-Sin, Park for their endless love and understanding and supporting.

Table of Contents

Chapter 1 ----- 6

Introduction

- 1.1 Purpose
- 1.2 Outline

Chapter 2 ----- 8

Background

- 2.1 GPS receiver components
 - 2.1.1 RF front end
 - 2.1.2 Baseband processor / correlator
 - 2.1.3 General purpose processor
- 2.2 Milestones of Open Source GPS development
 - 2.2.1 OSGPS
 - 2.2.2 ARMGPS
 - 2.2.3 GPL-GPS
 - 2.2.4 Namuru GPS
- 2.3 Open Source GPS receiver design
- 2.4 Namuru Open Source GPS project

Chapter 3 ----- 14

System Development

- 3.1 System Design Flow
- 3.2 SigNav MG5001 GPS receiver
 - 2.2.1 Porting GPL-GPS binary to SigNav MG5001 GPS receiver
- 3.3 Namuru GPS receiver
 - 2.3.1 FPGA (Reconfigurable computing)
 - 2.3.2 Namuru GPS tracking module
 - 2.3.3 soft-core processor (NiosII)

Chapter 4 ----- 19

Porting RTOS eCos

- 4.1 Real Time Operating System 'eCos'
- 4.2 Generating 'eCos' library

2.2.1 'nios2configtool'

Chapter 5 ----- 22

Transforming GPL-GPS

- 5.1 Migration from SigNav MG5001 to Namuru GPS
 - 5.1.1 Makefile
 - 5.1.2 Hardware IO access
 - 5.1.2.1 16bit hardware access functions
 - 5.1.2.2 32bit hardware access functions
 - 5.1.2.3 Channel control/accumulate functions
 - 5.1.2.4 Interrupt Accumulator
 - 5.1.3 Interrupt Service Routine
 - 5.1.4 Debugging
 - 5.1.4.1 LED control
 - 5.1.4.2 'diag_printf'
- 5.2 Improvement of functionality
 - 5.2.1 Clear Screen
 - 5.2.2 JTAG / UART and Circular Buffer Handling
 - 5.2.3 DISPLAY_CLEAR mode
 - 5.2.4 Extracting X,Y,Z log data
 - 5.2.5 Initial Location
- 5.3 GPL-GPS source code version update
 - 5.3.1 Comparison between OSGPS and GPL-GPS
 - 5.3.2 Update the latest GPL-GPS source code
 - 5.3.3 Identifying Bugs

Chapter 6 ----- 36

Verification and Testing

- 6.1 Verification of Positioning
 - 6.1.1 Screen capture of tracking satellites
 - 6.1.2 Trimble Planning
 - 6.1.2.1 Verification of visibility
 - 6.1.2.2 Verification of elevation
 - 6.1.2.3 Verification of azimuth
- 6.2 Static Testing

Chapter 7	-----	40
Conclusion & Future Work		
7.1	Conclusion	
7.2	Future Work	
Bibliography	-----	42
Appendix A	-----	
	Namuru FPGA GPS tracking module	
Appendix B	-----	
	Porting GPL-GPS to SigNavMG5001	
Appendix C	-----	
	Makefile for GPL-GPS	
Appendix D	-----	
	MATLAB script	
Appendix E	-----	
	TODO file for GPL-GPS	

Chapter 1 Introduction

1.1 Purpose

The purpose of this ‘Namuru Open Source GPS receiver project’ is developing open source firmware for the Namuru UNSW designed FPGA-based GPS receiver.

As a result of this project, complete open source GNSS research platform will be developed. When this project complete, the results of this work will be made available to the Global Navigation Satellite System (GNSS) research community alongside the Namuru receiver as a complete open source GNSS research platform.

The main task of this project is porting existing open source GPS software to a Namuru circuit board which is the NiosII soft core processor running on an Altera FPGA chip. The open source GPS firmware has previously been ported to the SigNav MG5001, using the 'eCos' RTOS.

The steps for the project followed below order:

- Orientation to the technology
- Investigation of the porting process
- Issues that need to be addressed.
- Core work
- Testing
- Documentation

1.2 Outline

This report aims to provide an understanding of developing open source GPS receiver and porting process of open source firmware towards ‘Namuru’ target hardware.

This report consists of seven chapters:

- **Chapter 1** is the introduction.
- **Chapter 2** talks in details about the background and milestones of open source GPS software development, especially GPL-GPS open source firmware together with open source GPS design philosophy.
- **Chapter 3** describes the system development environment of the design process, including the system design flow along with the porting steps of GPL-GPS firmware

to commercial GPS receiver, SigNav MG5001 as an porting exercise. And brief introduction to 'Namuru' UNSW designed FPGA-based GPS receiver.

- **Chapter 4** is general introduction of RTOS (Real Time Operating System) 'eCos'.
- **Chapter 5** focuses on main porting process of GPL-GPS firmware in general, migration to different hardware in particular. The porting process was stated in detail and it includes discussed issues, modifications, debugging as well as improvements.
- **Chapter 6** is about the verification of positioning result with Trimble Planning software. And this chapter also shows the testing method/result of the log data of the Namuru GPL-GPS receiver with plotting graph.
- **Chapter 7** briefly restates the initial goals of this project, the conclusions resulting from the output of this project and the discussion of possible enhancement and approaches for further development.
- All references to the related work of this project listed in Bibliography.

and five appendices:

- **Appendix A** includes Namuru FPGA GPS tracking module datasheet
- **Appendix B** includes the detailed step based instruction of porting GPL-GPS firmware to SigNav MG5001 receiver.
- **Appendix C** include Makefile written in order to compile GPL-GPS with GCC compiler.
- **Appendix D** include MATLAB script that used to generate plot graph for static testing result
- **Appendix E** include TODO file of GPL-GPS.

Chapter 2 Background

2.1 GPS receiver components

A conventional GPS receiver consists of three major components.

They are RF front-end, baseband processor, in other words correlator, general purpose processor.

2.1.1 RF front end

GPS receivers have a RF front end (radio frequency front end) chip that down converts and digitizes the 1.57542 Ghz carrier into a signal a few megahertz wide.

After down conversion, the resulting signal is over sampled by a two bit flash analog to digital converter at a sample rate of 5.7 Mhz. This digitized stream is sent to a correlator chip to be further processed.

2.1.2 Baseband processor / correlator

Correlator is a core component of GPS receiver. It is implemented in either software or hardware within a tracking channel in order to shift and compare the incoming signal from RF front end side. This correlator includes hardware such as Gold code generator, binary multipliers, and digitally controlled oscillators etc.

2.1.3 General purpose processor

GPS receiver requires a general - purpose processor to control the channel's DCO.

General - purpose processor converts raw data into tasks like maintaining receiver status.

2.2 Milestones of Open Source GPS development

There is a milestones of open source GPS development started from OSGPS project conducted by Clifford Kelly. And it becomes motivation of this Namuru Open Source GPS receiver project.

2.2.1 OSGPS

Dr. Clifford Kelly began working on open source GPS software around 1995. He called the project OpenSource GPS (OSGPS). By using a hacked commercial receiver based on the GP2021 chip, bypassing the receiver's microprocessor and connecting the hardware correlator chip directly to a 486 PC using an ISA prototyping board, OSGPS is able to run all its software on the host PC. In 2002, Kelly's work was published and it was the first open source GPS receiver software written.

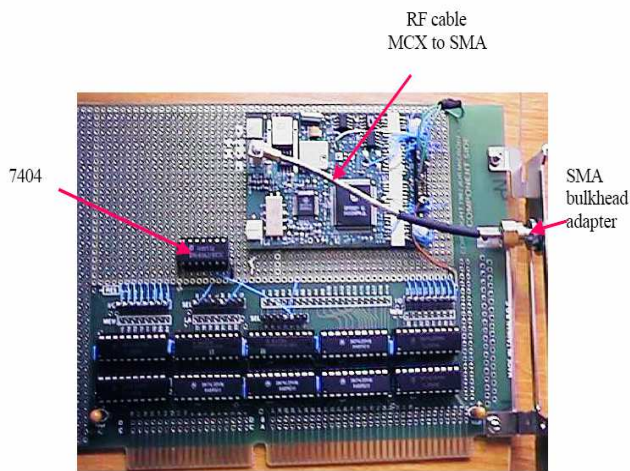


FIGURE 1. OSGPS GPS receiver board

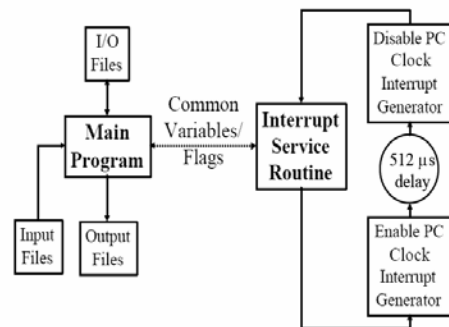


FIGURE 2. OSGPS software dataflow

2.2.2 ARMGPS

In 2004, Takuji Ebinuma started his project by taking Kelley's OSGPS code and porting it to a commercial receiver. Takuji's port ran on MG5001 hardware using the GP4020's built in bootloader. Tak started his project with OSGPS v1.15 in 2004. Tak stripped down the original code. He break down OSGP's main line task into thread based task. However, the implementation was not effective than original OSGPS code because the threads gets called by timer delay instead of event driven.

Furthermore, He attributed ARMGPS to the real time operating system 'JaysOS' which forced a primitive thread architecture with no inter-process communication (IPC) such as semaphores and locks. Therefore, while ARMGPS does works, it is unstable. Although there were several problems with the project, his work became the world's first open source stand-alone GPS receiver board. The project web site is available. [11]

2.2.3 GPL-GPS

In late 2004, Andrew Greenberg, a member of the Portland State Aerospace Society (PSAS) at Portland State University, started his project with taking Tak's code as a base and OSGPS v.1.17 as a reference. He restructured/replaced Tak's code almost 80% to write GPL-GPS code. The GPL-GPS's first position fix was on May 2nd, 2005. GPL-GPS supports the Zarlink GP4020, a 12-channel L1 C/A GPS receiver baseband processor with an integrated 32 bit ARM7TDMI microprocessor.

The GPL-GPS software currently runs on the ARM processor of the SigTec MG5001 GPS board using the eCos real-time operating system. GPL-GPS is intended to be part of an inexpensive 6 degree-of-freedom differential GPS-aided inertial navigation system designed for a small, inexpensive, and open source sounding rocket platform being developed by the Portland State Aerospace Society.



FIGURE 3. GPL-GPS receiver board

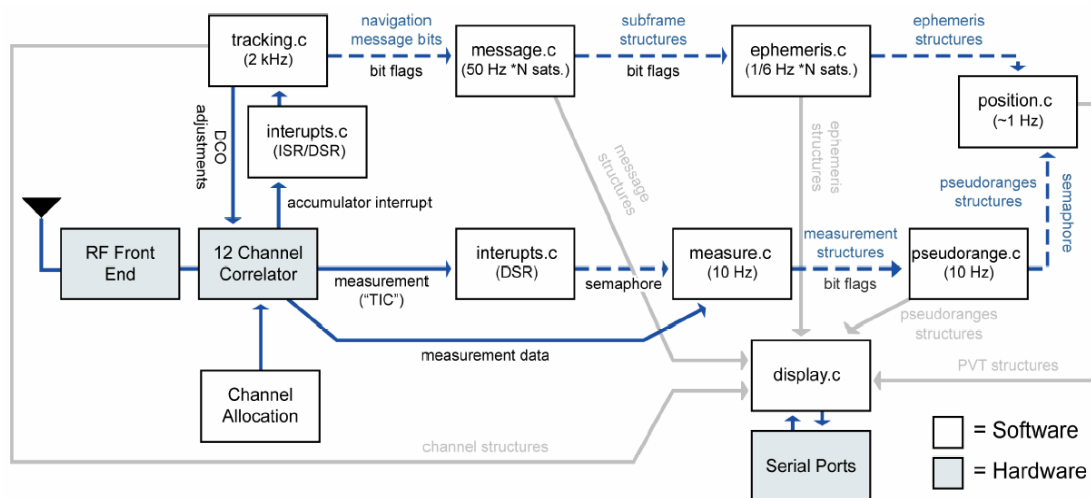


FIGURE 4. GPL-GPS tasks and flowchart

This GPL-GPS is being reworked. We hope to rapidly improve its performance so that it is comparable to similar commercial products. GPL-GPS is open source software based on GNU license. The detailed information of GPL-GPS is available on the web.[12]

2.2.4 Namuru GPS

The new FPGA based GPS L1 receiver has developed by UNSW and NICTA as a joint project. ‘Namuru’ means ‘to see the way’ in the language of the EORA people who inhabited an area around Sydney before the arrival of British.

The Mitel GPS Architect has been ported to this Namuru GPS receiver successfully after some tuning of the tracking loops. The GPS Architect employs a task-switching operating system. The various testing performed including dynamic testing. The detailed information of this porting process and static/dynamic test results are available ‘The Namuru Open GNSS Research receiver.’[1].

The detailed information regarding Namuru GPS receiver is illustrated in Chapter3.3.

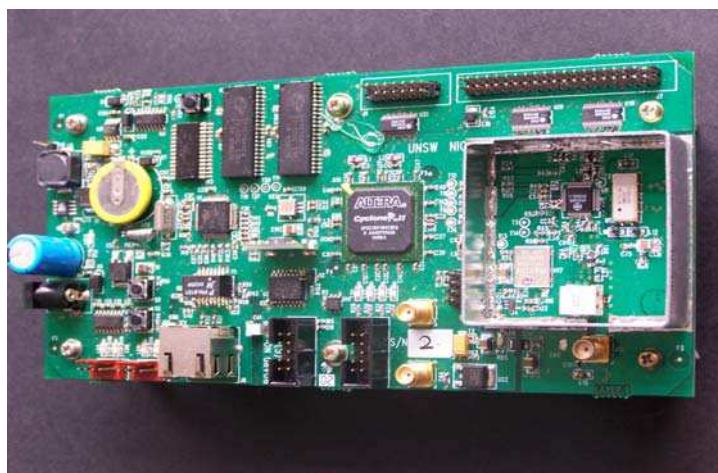


FIGURE 5. Namuru GPS receiver board

Summarization of Open Source of GPS projects

Project	Researcher	RTOS	Board
OSGPS	Clifford Kelly	N / A	x86 PC with Zarlink GP2021 correlator on PCI card
ARMGPS	Takuji Ebinuma	Jays OS	SigTec 5001 receiver
GPL - GPS	Andrew Greenberg	eCos	SigTec 5001 receiver
Namuru GPS	Peter Mumford	GPS Architect OS	Namuru GPS receiver

- SigTec 5001: Zarlink 4020 chip with ARM7TDMI processor + GP2015
- Namuru GPS receiver: Zarlink 2015 chip with NiosII soft-core processor on FPGA device

2.3 Open Source GPS receiver Design

GPL-GPS is based on the “open source” design philosophy. Andrew Greenberg who developed the GPL-GPS followed six guiding design principles based on the open source design philosophy. These design philosophy found in his paper ‘*Open Source Software for Commercial Off-the-Shelf GPS Receivers*’ [2]. Below is summarized form of his statements from his paper.

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

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

Meant for deeply embedded applications: The receiver should be usable in deeply embedded applications like sounding rockets and UAVs. This means attempting to minimize size, weight and power requirements...

Portable software: The software design should be as modular and portable as possible...

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

Available: All software and documentation must be made available on the internet. Any developer with modest experience who is willing to read the documentation should be able to quickly and easily start development...

The GNU General Public License (GPL) was chosen as the software license because of its general acceptance as an open source license.

2.4 Namuru Open Source GPS project

Developing a Open Source GPS receiver project for the Namuru UNSW designed FPGA-based GPS receiver is launched late 2006 at University of New South Wales in School of Surveying and Spatial Information Systems. This Namuru Open Source GPS project follows the above open source paradigm. All the hardware design of the Namuru GPS circuit board are available on the web.[15] And the GPL-GPS software will be ported to Namuru GPS board. And the eCos real-time operating system, which is also open source, will be ported to operate the system. In order to support developers, datasheet of GPS tracking module are available to public.

Therefore, when the GPL-GPS firmware is ported to Namuru GPS receiver together with eCos RTOS successfully, it became an open source GPS receiver in term of both hardware and software so that it can be distribute under The GNU General Public License agreement.

Chapter 3 System Development

3.1 System Design Flow

Primarily, NiosII IDE is used for this project in terms of GPS software development. Previously, hardware GPS receiver development project called ‘The UNSW/NICTA FPGA-based GPS receiver’[3] has been conducted. The ‘Namuru GPS receiver’ was designed by using Quartus II and SOPC Builder. The implemented hardware design was adopted for this Namuru OpenSource GPS receiver project. Therefore, QuartusII software was used in order to modify existing hardware design such as correlator, FIFO, UART etc.

QuartusII software

In this project, The ALTERA QuartusII design software was used. The QuartusII software provides a complete, multiplatform design environment that easily adapts to specific design needs.

SOPC Builder

The SOPC Builder, which is included with the QuartusII software, provides a standardized, graphical environment for creating SOPC designs composed of components such as NiosII processor family, memory interface, and various peripherals. With this SOPC Builder, each components and interfaces can be tailored and customized toward target hardware.

Nios II Integrated Development Environment

The Nios II IDE is the software development graphical user interface (GUI) for the Nios II processor. GPS software development task was accomplished with this software including editing, building, and debugging as well as simulating programs.

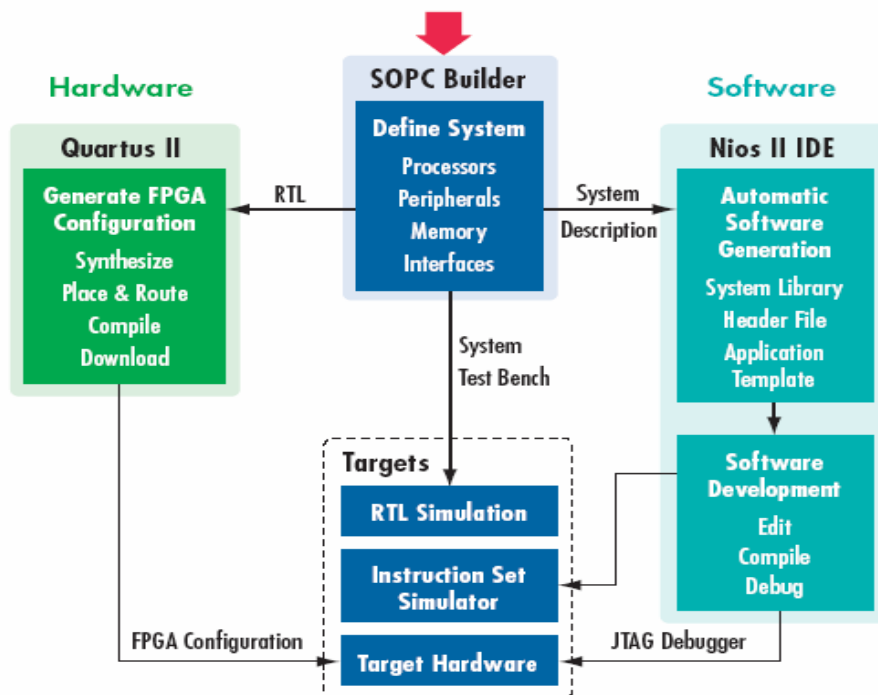


FIGURE 6. NiosII Embedded Processor Development Flow

3.2 Porting GPL-GPS to SigNav MG5001 GPS receiver

Prior to beginning of ‘Namuru OpenSource GPS receiver’ development, porting GPL-GPS software to SigNav MG5001 receiver task has been conducted for various reasons. The SigNav MG5001 is based on Zarlink's GP4020 GPS baseband processor, which combines a 12-channel correlator for L1 C/A code and a 32-bit ARM7TDMI core in a single package. The GPL-GPS software development files and instructions are available on the web. [15]

This task involve several sub-tasks:

- Installation of Cygwin software tools and eCos configuration tool
- Download the latest arm-elf toolchain
- Check out the latest eCos source code out of CVS directory
- Download the latest GPL-GPS software from the web
- Setting the eCos configtool
- Burn RedBoot boot loader image into the SigNav MG5001’s flash

The output of this tasks gives:

- Porting experience with specific hardware
- Expected output with display when GPL-GPS software is successfully run.
- Counter reference to Namuru OpenSource GPS receiver which will also run GPL-GPS software.
- Getting familiar with eCos and eCos configuration tool

RedBoot is a bootstrap environment for embedded systems. It is a part of the royalty-free embedded real-time operating system eCos. eCos is distributed by RedHat, and all sources are delivered in open source. RedBoot binaries is also available from the website which will allow skip some of tasks above. The detailed step by step tasks performed is available in **Appendix B** with screen-captured images.

3.3 Namuru FPGA based GPS receiver platform

The Namuru GPS receiver has an architecture that the base-band processor and general-purpose processor combined into one FPGA and RF front-end chip is interface to this FPGA chip. This Namuru GPS receiver currently support L1 signal only due to limitations to the frequency plan of GP2015 chip.[4] Below diagram illustrate the basic structure of Namuru GPS receiver.

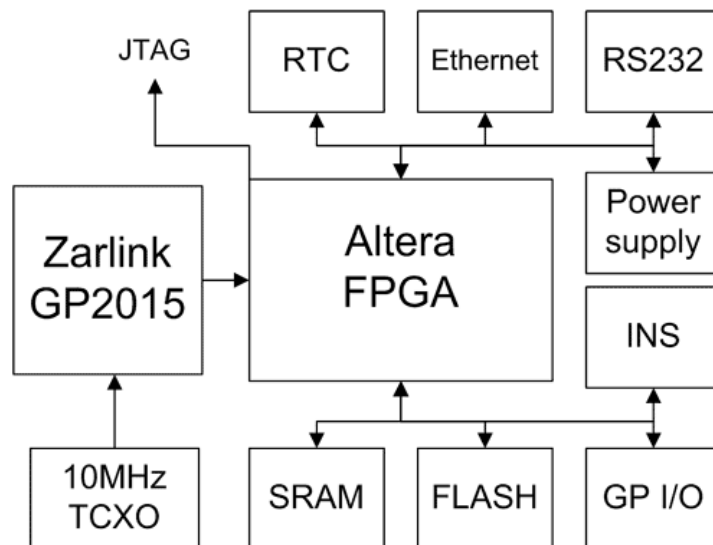


FIGURE 7. structure diagram of Namuru GPS receiver

3.3.1 FPGA (Field Programmable Gate Arrays)

A field programmable gate array (FPGA) is a semiconductor device containing programmable logic components and programmable interconnects from the definition of Wikipedia encyclopedia. FPGAs technology allows reconfigurability, where the function of the circuit can be dynamically changed and reloaded. This enable to modify the system without breaking it therefore provide flexibility of the system development. The 'EPM2C35' device was selected from Altera CycloneII FPGA family for this GPS platform. This device provides 33,216 LEs (logic elements) and 105 memory blocks. There is a strong case for using an FPGA for GPS receiver design because of ability to easily reconfigure the digital processing environment. According to the page, 'The UNSW/NICTA FPGA-based GPS receiver: A tool for GNSS research'[3], with many new signals arriving over the next decade this gives the flexibility of design.

3.3.2 RF front end : Zarlink 2015 chip

The Zarlink GP2015 RF front end chip is currently used in commercial GPS receivers. It was conceived by GEC Plessey as part of their GPS chip set, then became a Mitel Semiconductor product and is now available from Zarlink Semiconductors. This RF front end chip convert GPS signals to suitable IF to be able to processed by base-band processor. A Zarlink 2015 chip is used along with filter circuits such as LNA (Low Noise Amplifier), TCXO (Temperature Compensated Crystal Oscillator).

3.3.3 GPS tracking module (correlator)

The GPS tracking module is a baseband processor/correlator of Namuru GPS receiver. It provides tracking and manipulation of signals from RF front end. This design is developed with both VHDL and Verilog. There are 12 channels and each channel has three complex code correlators.

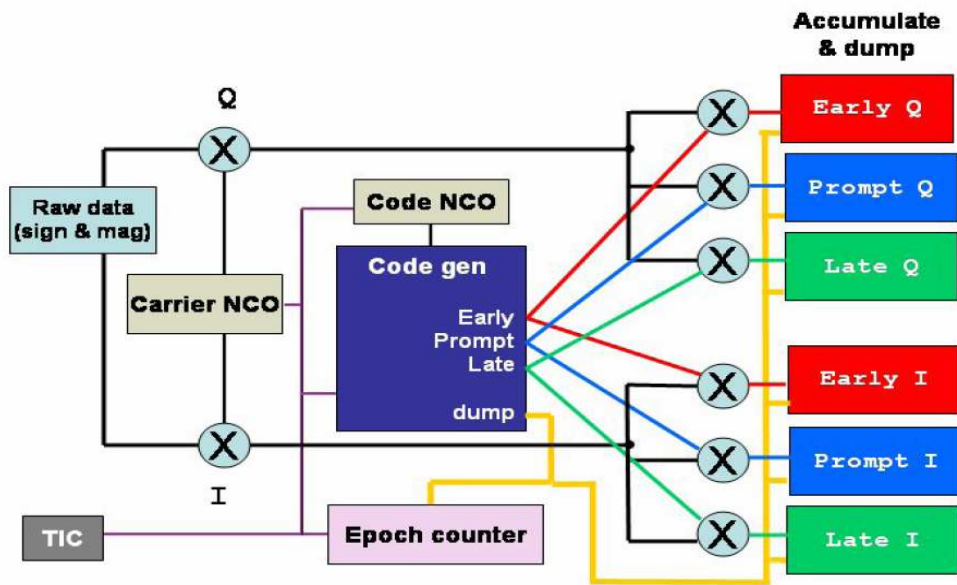


FIGURE 8. tracking channel block diagram

The GPS tracking module is connected to NiosII soft-core CPU as a memory –mapped peripheral via the Altera Avalon bus for further processing of data.

3.3.4 NiosII soft-core processor

The NiosII processor is a configurable soft-core processor, as opposed to a fixed, off-the-shelf microcontroller. Configurable means that features can be added or removed on a system-by-system basis to achieve purposes. Soft-core means the CPU core is offered in soft design form rather than fixed in silicon, and it can be targeted to Altera FPGA devices. Soft-core processors can migrate immediately to the latest FPGA technology.

The Altera Nios II family of embedded processors currently consists of three processor cores (fast, standard, economic) that implement a common instruction set architecture, each optimized for a specific price/performance point, and all supported by the same software tool chain. Among these cores, NiosII/ fast core is currently integrated into the FPGA chip of Namuru GPS receiver. The Nios II/f “fast” core is designed for high execution performance by maximizing the instructions-per-cycle and execution efficiency

Chapter 4 Porting RTOS eCos

4.1 Real Time Operating System ‘eCos’

‘eCos’ is a real time operating system targeted to embedded systems. ‘eCos’ stand for ‘embedded Configurable operating system’. It is a royalty-free open-source.

‘eCos’ is not a standard real time operating system that runs independently on the target system and provide an environment to run applications rather is a runtime system which a static pre-compiled library that the application links against during compile time. The ‘eCos’ library provide all of the functions of a standard operating system such as startup, real-time operating system kernel, scheduler etc.

One of the key aspects of ‘eCos’ is its configuration system. It allows control and configuration according to the system requirements. ‘eCos’ uses a hardware abstraction layer (HAL) library to keep its kernel independent of the underlying system architecture. Therefore, the kernel can be almost completely independent of the hardware. The hardware abstraction layer can be broken down into three sub modules: architecture, variant, platform.

For this project, ‘eCos’ will provide requirements of a RTOS such as hard real time performance. Therefore, it will be ported to Namuru GPS receiver

4.2 Generating ‘eCos’ library

4.2.1 ‘nios2configtool’

‘nios2configtool’ is a configuration tool of the ‘eCos’ operating system for the NiosII processor. The ‘eCos’ operating system provides an effective software development platform for NiosII soft-core processor. The NiosII port of ‘eCos’ is similar to other ‘eCos’ architecture except that no platform or variant port is required for NiosII systems. The hardware parameters and device drivers are automatically configured based the SOPC Builder system configuration. It is possible to build ‘eCos’ applications from within the NiosII IDE using an ‘Advanced C/C++ Project’.

During this project,

- eCos source was check out from CVS directory
- Install the ‘nios2ecosconfigtool’ software
- eCos library directory was configured and created based on the configuration file that is generated from SOPC Builder.
- The generated eCos library was imported to the current project by using NiosII IDE.

This ‘nios2configtool’ was invoked from a NiosII SDK shell.

For example, if the target configuration file generated from SOPC Builder is ‘nios_cpu’ and the name of cpu is ‘cpu’, then it is invoked by below command line.

> nios2configtool -- ptf=<SOPC Builder PTF file>/nios_cpu --cpu=cpu

The screen capture image is available at next page.

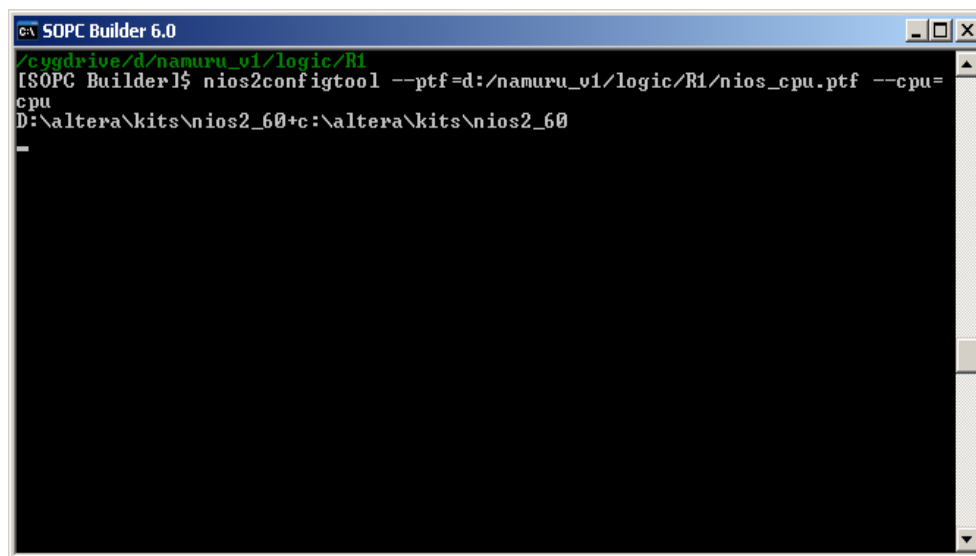


FIGURE 9. Screen capture of invoking nios2configtool

After run the ‘configtool’, open the existing configuration file called ‘ecos_a.ecc’ to re-configure for current project. The screen captured was presented below.

After configuration, clicking ‘Build’ will generate configured ‘eCos’ library.

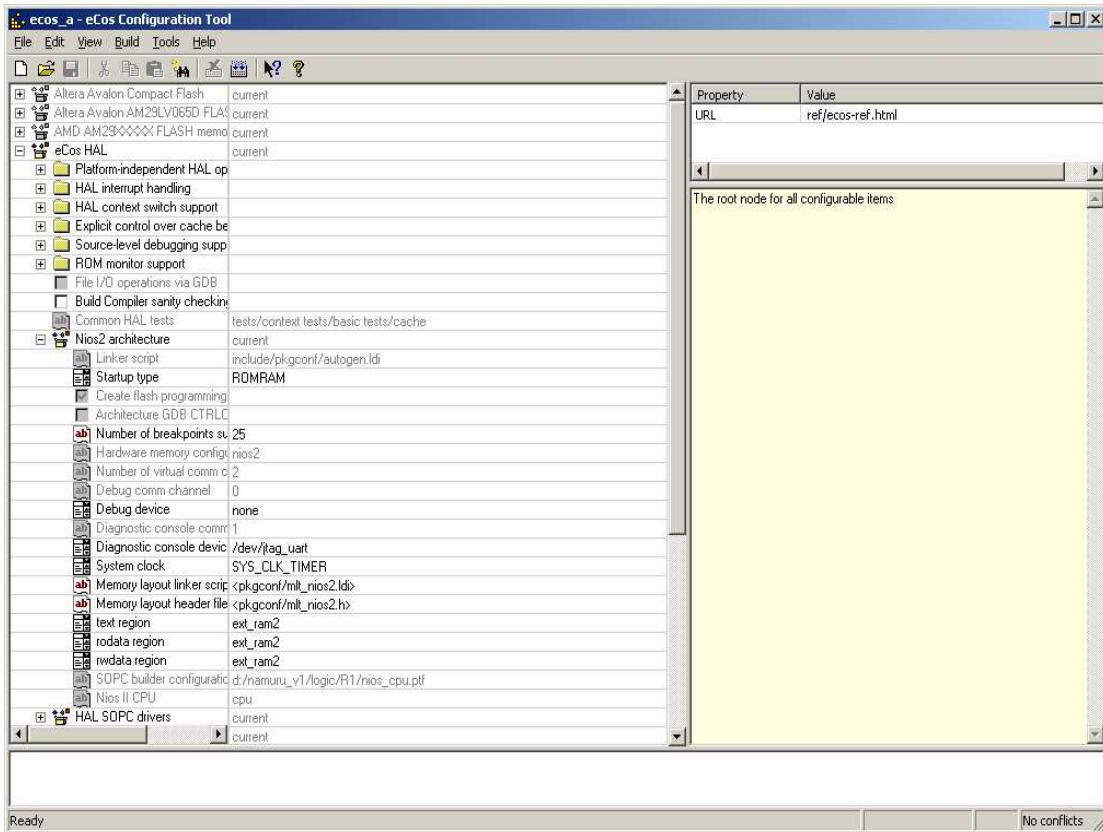


FIGURE 10. Screen capture of ecos configtool

Chapter 5 Transforming GPL-GPS

5.1 Migration from SigNav MG5001 to Namuru GPS

From the GPL-GPS website [15], the latest GPL-GPS source code('gpl-gps_2005-05-11.tgz') was downloaded. It was extracted and imported to as a NiosII IDE along with creating a new 'Advanced C/C++ project'.

5.1.1 'Makefile'

As the NiosII IDE 'Advanced C/C++ project' requires managing own Makefile in order to compile the current project files, writing a Makefile was first task.

After successfully compile with written Makefile, modification of source code applied while the project move on in order to keep clean compile. The written Makefile is attached in Appendix C.

5.1.2 Hardware I/O access

Identifying hardware I/O (input/output) access functions and predefine values for specific hardware platform (SigNav MG5001) from GPL-GPS code was performed next. This hardware related codes are located in 'gp4020.h', 'gp4020_io.h', 'gp4020_io.c'. Instead of modifying these functions, new files 'namuru_corraddr.h', 'namuru_io.h', 'namuru_io.c' were written/imported to replace them respectively. These new files include defined constant values and hardware related functions in associate with 'Namuru GPS datasheet'[5]. And 'SYSTEM_BUS_WIDTH' was defined at 'namuru_io.h' for compilation. Including '#include <cyg/hal/io.h>' to 'namuru_io.h' requires to define this variable.

```
#define SYSTEM_BUS_WIDTH 32
```

The identified hardware I/O access functions are categorized into four groups. And further details discussed below.

- 16bit hardware access functions
- 32bit hardware access functions

- Channel control/accumulate functions
- Interrupt accumulator related functions

5.1.2.1 16bit hardware access functions

Originally, 16bit hardware access functions are written in order to support 16bit integer type since ‘SigNav MG5001’ uses 16bit correlator which support 16 bit integer types. It is supported from Andrew Greenberg’s thesis paper.[6] where

5.3.2 Optimizing for GPL-GPS

GP4020-friendly variable types: Since the correlator uses 16 bit integers, much of the integer processing can be moved from 32 bit to 16 bit words to speed up access to the external 16 bit SRAM. Because of the ARM7TDMI’s single-cycle 32,16 and 8 bit memory access abilities this should not slow down processing.

...

Since the Namuru GPS receiver uses NiosII soft-core processor that doesn’t support 16 bit memory processing, it is not necessary to re-implement these functions. Therefore, they have excluded from new file. And there is one power control related hardware access macros.

line 87, ‘gpl-gps.c’
out16(GP4020_CORR_RESET_CONTROL, reset_control_shadow);

This power control part was commented out due to Namuru GPS receiver doesn’t support power control yet.

5.1.2.2 32bit hardware access functions

32bit hardware access functions were written to replace existing functions so that it is able to communicate with Namuru GPS receiver hardware. After the discussion, minor modification of original source code was encouraged. Hence, the name and input/output type of values for each functions remains same but the piece of code inside was changed. For example, existing function ‘out32’

```
inline void out32 (cyg_uint32 address, cyg_uint32 data) {
    *((volatile cyg_uint32 *)address) = data;
}
```

was modified like below

```
inline void out32 (cyg_uint32 address, cyg_uint32 data) {
    write_to_correlator(address, data);
}
```

where ‘write_to_correlator(address, data)’ was defined as

```
#define write_to_correlator(A, D) IOWR(GPS_TRACKING_BASE,A,D)
```

And the MPC Area setting for ARM processor was commented out

```
line95 ,gpl-gps.c
// First, make sure the MPC Area 3 is set correctly:
// out32( GPS4020_MPC_AREA_1, GPS4020_MPC_A1_DEF);
// out32( GPS4020_MPC_AREA_2, GPS4020_MPC_A2_DEF);
// out32( GPS4020_MPC_AREA_3, GPS4020_MPC_A3_DEF);
// out32( GPS4020_MPC_AREA_4, GPS4020_MPC_A4_DEF);
```

5.1.2.3 Channel control/accumulate functions

There are three channel related functions inside of GPL-GPS software. They are ‘_gp4020_channel_control’ and ‘_gp4020_channel_accumulators’. It is obvious that channel control/accumulator is related to correlator. And SigNav MG5001 receiver uses different correlator from Namuru receiver. Therefore this part requires rewrite from beginning unlike above functions. Peter wrote new lines of code to replace them. For instance,

```
volatile union
    _gp4020_channel_control *channel_control =
(volatile union _gp4020_channel_control *)
    GP4020_CORR_CHANNEL_CONTROL;
```


This has been replaced with:

```
write_to_correlator( (ch * CH_BASE_STEP + PRN_KEY), PrnCode[prn]);
```

While configuring the GPL-GPS course code for Namuru GPS hardware platform, 'tracking.c' and 'allocate.c' were considerably modified due to correlator related function. This modification/restructuring of tracking/allocating part of source code has accomplished by Peter.

5.1.2.4 Interrupt Accumulator

Interrupt signal accumulator is another part of hardware related part.

Inside of 'gpl-gps.c' function, CYGNUM_HAL_INTERRUPT_CORR_ACCUM was replaced with GPS_TRACKING_IRQ which have an hardware address of the GPS Tracking Module (coorelator) connected to.

The 'status' and 'new_data' registers are implemented in the Namuru hardware and 'accum_status_a' and 'accum_status_b' were replaced by 'status', 'new_data' respectively based on 'Namuru FPGA GPS tracking module' datasheet.'[5]

5.1.3 Interrupt Service Routine

Interrupt service routine was initialized by calling 'initialize_namuru_interrupts'. 'ACCUM_INT' rate was programmed as 0.5 ms and 'TIC rate' was initialized as 100 ms as default.

```
// program ACCUM_INT rate  
out32( PROG_ACCUM_INT, 0x00004E1F ); //0.5ms default (pjm)  
// program TIC rate  
out32( PROG_TIC, 0x3D08FF ); //100ms default (pjm)
```

The TIC signal synchronizes measurements across all channels. The accumulator interrupt provides an interrupt to the microprocessor so that accumulation and measurement data can be collected and processed.

By using LED signal, the interrupt rate was confirmed. LED1 set to toggle whenever the interrupt occurs. And the oscilloscope set up to the LED1 to get interrupt signal frequency. The interrupt signal was occur every 0.5 milliseconds.

5.1.4 Debugging

One of the challenging part of this project was debugging. Especially, debugging multi-threaded applications can be challenging because it involves a debugging deadlocks. In multi-threaded applications, timing is everything. Any change in the application or runtime environment (running under the debugger) can cause application timing to be significantly altered which can cause a very consistent deadlock to not occur when you are attempting to investigate the issue.

Most popular and simplest way of debugging in embedded system is using LED. Turning on/off LED is simply writing to specific address of the hardware. Hence it doesn't involve complex system calls. Namuru GPS receiver supports four LEDs.

5.1.4.1 LED control

In order to manipulate LEDs in Namuru receiver, new files 'led.h' and 'led.c' were written. Four LED relate functions were written to operates turning on/off and toggling. For example, to turn on certain LED 'led_turnon' was called

```
void led_turnon ( unsigned short led ){  
    led_status |= led;  
    write_to_led(led_status);  
    return;  
}
```

where 'write_to_led' is defined as below

```
#define write_to_led(D) IOWR(LED_PIO_BASE,0,D)
```

Each LED was used for different purpose.

- LED0 used for heartbeat driven from ISR/DSR
- LED1 used for interrupt
- LED3 used for each tasks in software
- LED4 used for general purpose

5.1.4.2 ‘diag_printf’ statement

‘printf’ statement is one of popular method for debugging however it does not always work right because data is not always printed synchronously with the call to ‘printf’.

Rather, printf buffers printed characters in memory and writes the output only when it has accumulated enough to justify the cost of invoking the operating system's write system call to put the output on your screen.

Therefore, ‘diag_printf’ was introduced.

According to eCos Kernel Architect expert, Nick Garnett

‘diag_printf’ is a low level routine that prints directly to the debug channel. It doesn't go through the C library, and can be used in DSRs and ISRs where C library calls would cause problems. It is used for tracing and assertions, and for system level diagnostics, where the presence of the C library cannot be assumed.

‘diag_printf’ is not real-time. It disables interrupts while doing its printing. So if a low priority function doing a ‘diag_printf’ it will stop a high priority function taking over when it becomes runnable. This can be really bad when you are using a slow serial port. It can give you some huge latencies. ‘diag_printf’ also has a few less modifiers and types implemented.

5.2 Improvement

5.2.1 JTAG UART and Circular Buffer Handling

JTAG UART was combined into top system. JTAG(Joint Test Action Group) is a port for downloading hardware logic and for running and debugging of the hardware. JTAG is also being used for the console connection.

Namuru GPS receiver support two UART ports, J3 and J4.

These ports were used for standard hyper-terminal application over the serial RS232 link for the displaying and monitoring purpose. These serial links was connected to another host computer where the hyper-terminal application is running.

Below diagram illustrates these connections.

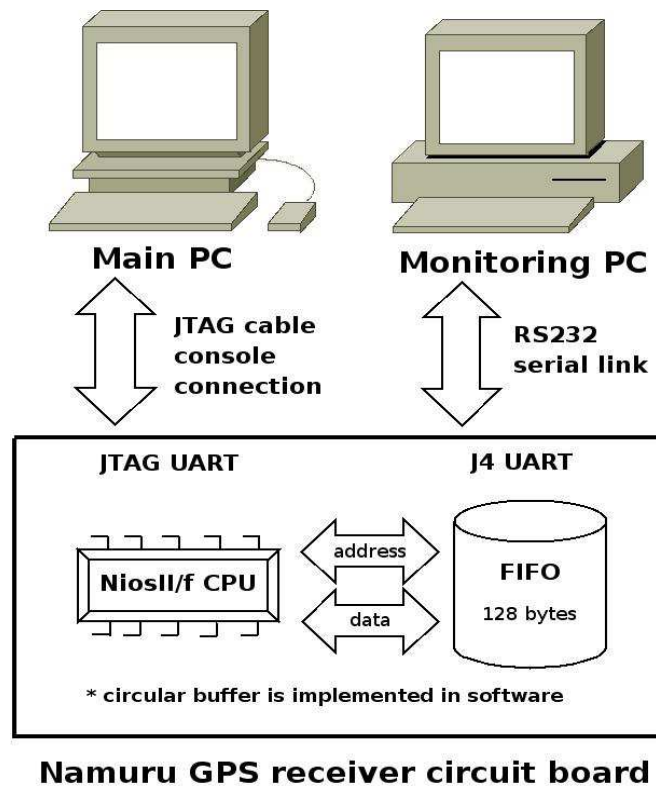


FIGURE 11. development communication structure

There is a hardware FIFO (First In – First Out) implementation in J4, which has the size of 128 bytes initially. However, the capacity of FIFO was not big enough to be capable of transmitting data for display from the GPL-GPS software. Therefore, expanding the size of FIFO attempted. However, it caused long time compilation process. Instead of expanding the capacity of FIFO, implementing circular buffer was suggested. From the previous project, circular buffer handling implementation was migrated to this project. The FIFO/circular buffer implementation consists of three functions. Prototype of three functions declared at ‘serial.h’ as below:

```
void uart2_initialize(void)
void uart2_transmit_string(char *string)
void tx_uart(void)
```

There is a status bit to check if the FIFO is empty. By using this signal, whenever the FIFO is empty, the data in circular buffer will feed next 128 bytes of data.

'tx_uart' was located in interrupt routine function called 'accum_dsr' which is the delayed service routine for the accumulator interrupt (accum_int). It gets called by the scheduler right after the ISR. Therefore, accum_dsr will get called at the same rate.

It will quickly enough to flush data in side of FIFO to prevent from losing data due to dropping information.

5.2.2 DISPLAY_CLEAR mode

Originally, GPL-GPS software support several display mode according to input character from standard input. The supported display modes are:

- DISPLAY_TRACKING correspond input command is 't'
- DISPLAY_MESSAGES correspond input command is 'm'
- DISPLAY_STOP correspond input command is 's'
- DISPLAY_EPHEMERIS correspond input command is 'e'
- DISPLAY_PSEUDORANGE correspond input command is 'r'
- DISPLAY_POSITION correspond input command is 'p'
- DISPLAY_DEBUG correspond input command is 'd'
- DISPLAY_LOG correspond input command is 'l'

There was a problem where displaying log after displaying tracking or some other mode. Since the HyperTerminal doesn't support clear screen function, it was expected to support 'DISPLAY_CLEAR' mode which clear all characters on the HyperTerminal window and then resume the last display mode. Hence 'DISPLAY_CLEAR' mode was implemented with correspond character 'c'.

- DISPLAY_CLEAR correspond input command is 'c'

5.2.3 Extracting X,Y,Z log data

After the Namuru GPS receiver runs GPL-GPS software and track a number of satellites, it is desired to support another display mode which display only X, Y, Z information of positioning data so that it can be collected for a while. As a consequence, the collected data can feed into MATLAB to generate plotting graph for the purpose of performance and error checking.

Therefore, another display mode was added called

- DISPLAY_PLOT correspond input command is 'h'

By typing 'h' character from host pc, the receiver will send X,Y,Z information as well as positioning flag. Below is example format is each information is divide by empty space and the order is X value, Y value, Z value and positioning flag. If the positioning flag is set to 1, it means it is positioning now.

```
-4.646427e+06 2.547824e+06 -3.537475e+06 0
-4.644596e+06 2.549812e+06 -3.538953e+06 1
...
```

In order to collect this x,y,z log data and save it to file, ComTerm is used. ComTerm is a serial communication program, not unlike the Windows standard program, Hyperterminal. ComTerm is intended for use with GPS receivers connected to any serial communication port on the computer. It supports 'LogtoFile' option which save data into file.

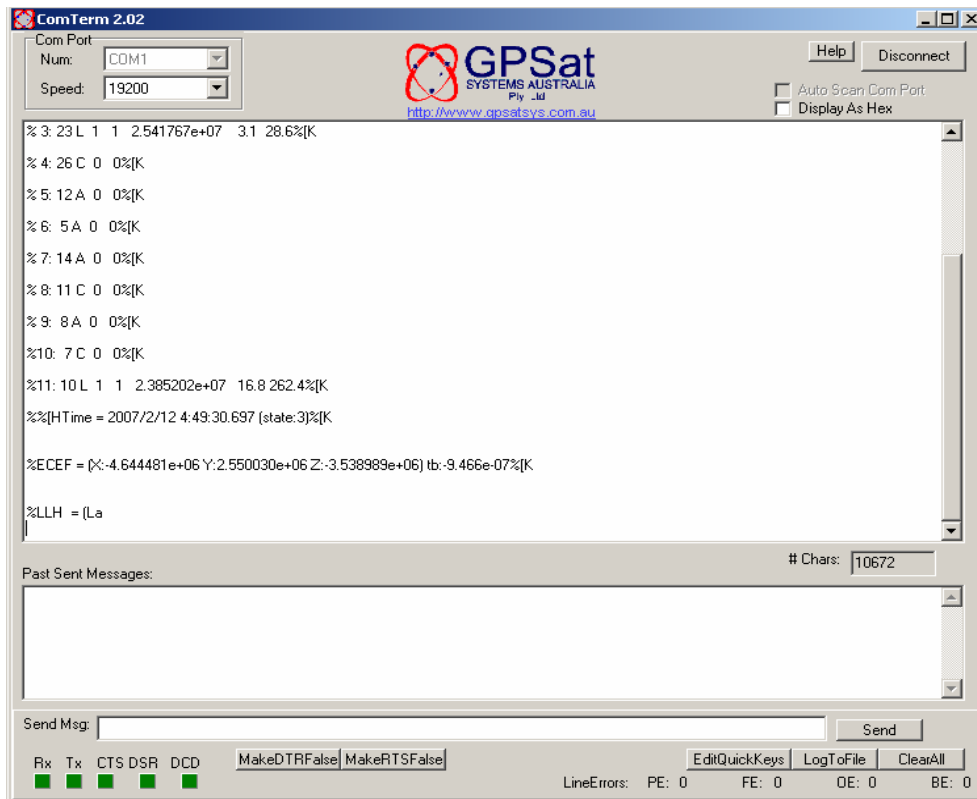
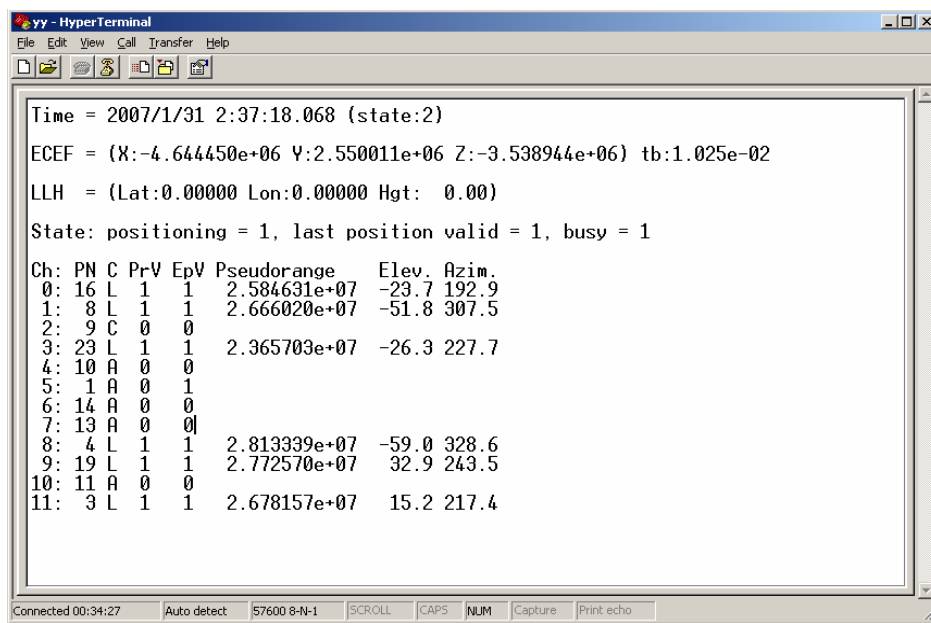


FIGURE 12. Screen capture of ComTerm

5.2.4 Initial Location

After displaying tracking of satellites, it is observed that the value of 'Elev' is negative. 'Elev' stands for elevation of the satellites and it is supposed to be positive values.



```
yy - HyperTerminal
File Edit View Call Transfer Help

Time = 2007/1/31 2:37:18.068 (state:2)
ECEF = (X:-4.644450e+06 Y:2.550011e+06 Z:-3.538944e+06) tb:1.025e-02
LLH = (Lat:0.00000 Lon:0.00000 Hgt: 0.00)
State: positioning = 1, last position valid = 1, busy = 1

Ch: PN C PrV EpV Pseudorange Elev Azim.
0: 16 L 1 1 2.584631e+07 -23.7 192.9
1: 8 L 1 1 2.666020e+07 -51.8 307.5
2: 9 C 0 0
3: 23 L 1 1 2.365703e+07 -26.3 227.7
4: 10 A 0 0
5: 1 A 0 1
6: 14 A 0 0
7: 13 A 0 0
8: 4 L 1 1 2.813339e+07 -59.0 328.6
9: 19 L 1 1 2.772570e+07 32.9 243.5
10: 11 A 0 0
11: 3 L 1 1 2.678157e+07 15.2 217.4

Connected 00:34:27 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print: echo
```

FIGURE 13. Screen capture of positioning display for original GPL-GPS source

While investigating the source code, below defined values was found inside of function 'satellite_azel' which calculates elevation and azimuth values for each satellite given a reference position.

```
#define north_x 0.385002966431406
#define north_y 0.60143634005935
#define north_z 0.70003360254707
#define east_x 0.842218174688889
#define east_y -0.539136852963804
#define east_z 0
#define up_x -0.377413913446142
#define up_y -0.589581022958081
#define up_z 0.71410990421991
```

It is suspected that these values are fixed value of certain position. And it is caused to generate negative elevation values. Therefore, modification of this part carried out.

Firstly, instead of using define values, they were declared as global variables.

Secondly, below piece of code was inserted which calculates the values of the global variable from given latitude and longitude values which is available when the receiver start to positioning.

```
if( receiver_pvt.valid){          // start to positioning
    north_x = -cos (receiver_llh.lon) * sin (receiver_llh.lat);
    north_y = -sin (receiver_llh.lon) * sin (receiver_llh.lat);
    north_z = cos (receiver_llh.lat);
    east_x = -sin (receiver_llh.lon);
    east_y = cos (receiver_llh.lon);
    east_z=0.0;
    up_x = cos (receiver_llh.lon) * cos (receiver_llh.lat);
    up_y = sin (receiver_llh.lon) * cos (receiver_llh.lat);
    up_z = sin (receiver_llh.lat);
}
```

After this modification, the Namuru GPS receiver start to display positive elevation values. The verification of correct satellite number and correct elevation value has performed for confirmation by using *Planning* software from *Trimble Pty.Ltd* provide. This will be illustrated in detail later in **CHATER 6**.

Next, although it fixed the problem of elevation, it still starts with incorrect values until the receiver track 4 satellites for positioning. Hence, ‘default_positioning’ function was implemented. It gets called only once when the GPL-GPS software start to run. It calculates a correct X,Y,Z coordinate for current position for the purpose of speed optimization

```
void default_positioning(void){

    north_x = -cos (HERE_LONG) * sin (HERE_LAT);
    north_y = -sin (HERE_LONG) * sin (HERE_LAT);
    north_z = cos (HERE_LAT);
    east_x = -sin (HERE_LONG);
    east_y = cos (HERE_LONG);
    east_z=0.0;
```



```

    up_x = cos (HERE_LONG) * cos (HERE_LAT);
    up_y = sin (HERE_LONG) * cos (HERE_LAT);
    up_z = sin (HERE_LAT);

}

```

The reference value of HERE_LONG and HERE_LAT is taken from file 'here.h' which include local information of current location.

```

// Location information for current Lab
// School of Surveying and Spatial Information System,
// University of New South Wales, Australia.

// X, Y, Z coordinatesa
#define HERE_X -4644463
#define HERE_Y 2549957
#define HERE_Z -3538929

// Latitude and Longitude ( Radian values )
#define HERE_LAT -0.5919633
#define HERE_LONG 2.6394931

```

X,Y,Z coordinate of current location was:

```
X : -4.644463, Y:2.549957, Z: -3.538929
```

The HERE_LAT and HERE_LONG values are radian values, which calculated from degree values HERE_LAT -33.917 and HERE_LONG 151.232.

5.3 GPL-GPS source code version update

Namuru GPL-GPS project started by downloading GPL-GPS source code from the web [15]. The GPL-GPS software was ported to Namuru GPS receiver successfully, although it seems that there are many bugs in it. And the behaviour of the software, which ported to Namuru GPS receiver, is not very reliable.

After the discussion, it is decided that intensive debugging is necessary to refine the Namuru GPS receiver and one of the refinement strategy of the receiver was revisiting original open source GPS softwares was suggested. The updated bugs from those projects can be easily applied to Namuru GPL-GPS project. Eventually, the updating known bugs may improve the performance of the Namuru GPS receiver.

Two open source GPS projects were visited, OSGPS and GPL-GPS.

5.3.1 Comparison between OSGPS and GPL-GPS

According to milestones of open source GPS software development, GPL-GPS was originally adopted from OSGPS source code that Clifford Kelly developed. As the OSGPS software is origin of the GPL-GPS software, the investigation of finding similarity between them has carried.

It is observed that the main structure of the OSGPS is completely different from the GPL-GPS software, although both contain same idea. One of the main differences is that the OSGPS uses a mainline/interrupt structure. The main program is driven by a timer interrupt on the host PC and all time dependent functions are called directly from a single interrupt whereas GPL-GPS uses event-driven threads. Each GPS task has divided into modules in GPL-GPS. As a result of this, it is concluded that it's hard to find similarity between OSGPS and GPL-GPS. As a consequence, applying improvement in OSGPS to GPL-GPS has declined due to short period of project. Instead updating the released version of GPL-GPS source code with up-to-date GPL-GPS source code was recommended.

5.3.2 Update the latest GPL-GPS source code

Since the downloaded GPL-GPS software which is released on 2005-05-11, it is discussed that there must be updates and improvements of it. Hence, the PSAS (Portland State Aerospace Society at Portland State University) server was visited in order to check out up-to-date GPL-GPS code from PSAS's repository.

After downloaded new version of GPL-GPS source code, the comparison with the released version was performed. Generally, there are no much differences between them. There are slight modifications that will not affect to Namuru GPL-GPS project such as target hardware related modifications and code optimizations.

Major changes observed at 'ephimeirs.c' in new version of GPL-GPS. Mostly, the processing function for sub-frame 1,2,3 was modified and for sub-frame 4 and 5 were

excluded. Since GPL-GPS doesn't support decoding almanac, this exclusion will not affect to current GPL-GPS software. It remains as future development. And one extra function called 'update_ephemeris' was added. There is a Andrew's statement inside of the source code regarding this new function:

```
/* According to the GPS documentation, the ephemerides from subframes 1,2,3  
are to be used as a complete set. This function tests the navigation messages to  
see if we can get a valid ephemeris from them.*/
```

The detailed comparisons between release version and new version is attached in **Appendix C**.

5.3.3 Identifying Bugs

Comment out function 'display_debug'

Function 'static void display_debug(void)' was commented out. Because this function gets called when CYGFUN_KERNEL_THREADS_STACK_MEASUREMENT option is enable. And there is a still unresolved bug with this option, so you might get a problem with this flag instead of help with following of stack size.

Description of problem:

Enabling CYGFUN_KERNEL_THREADS_STACK_MEASUREMENT causes the stack frames to confuse GDB.

Refer to related discussion on web site :

<http://sourceware.org/ml/ecos-discuss/2004-05/msg00239.html>

http://bugs.ecos.sourceware.org/show_bug.cgi?id=84530

Chapter 6 Verification and Testing

6.1 Verification of Positioning

The Namuru GPS receiver was connected to an antenna mounted on the roof of the EE Building at UNSW. And the ported GPL-GPS software ran on the Namuru GPS circuit board. After for a while, the receiver started to track satellites. Once it is positioning, which means the receiver start to track four satellites, the screen was captured for verification purpose.

6.1.1 Screen capture of tracking satellites

Example of Positioning on Namuru GPS receiver

Date / Time 30th/Jan /2007, 14:46
Software GPL-GPS software
GPS receiver Namuru GPS receiver

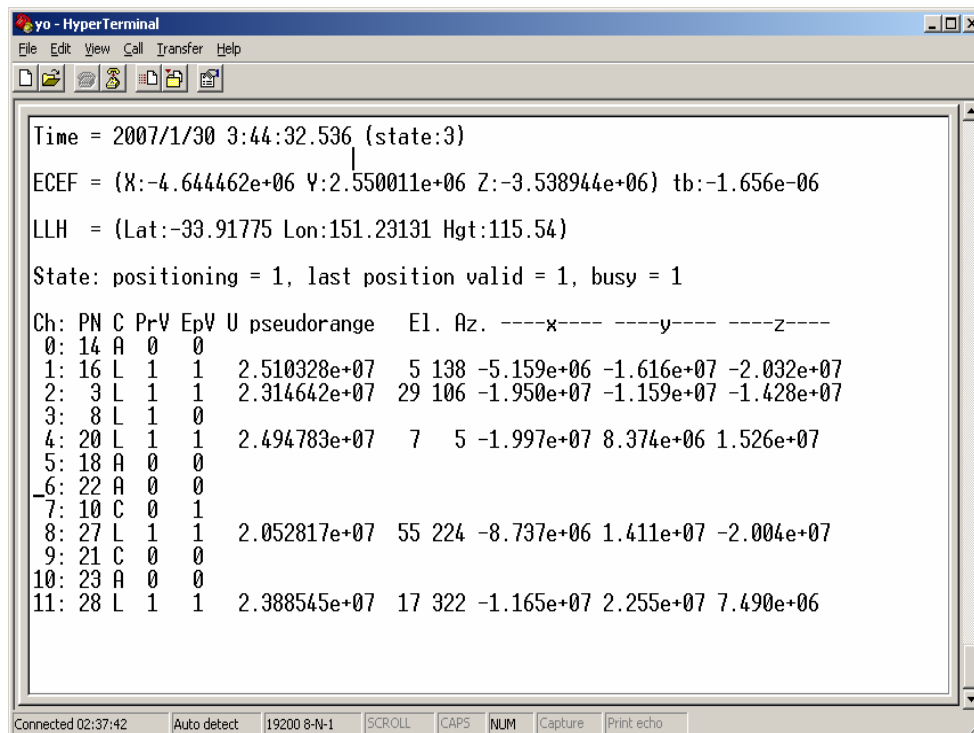


FIGURE 14. Screen capture of positioning display for updated GPL-GPS

Location information of S.S.I.S lab

X, Y, Z coordinates

X -4644463 Y 2549957 Z -3538929

Latitude and Longitude (Degree values)

Latitude : -33.917 degree / -0.5919633 radian

Longitude : 151.232 degree / 2.6394931 radian

6.1.2 Trimble Planning

Trimble's Planning software is a stand-alone software supporting any form of analysis to determine visibility for GPS, GLONASS, IGSO and geostationary satellites.

The Trimble 'Planning' software was used to display visibility, elevation, sky plot and world projection of selected satellites graphically. In order to perform this task, the latest ephemeris file was downloaded from the web. And parameters for stations such as location, time span and time zone, obstructions were defined.

More information of this software is available on the web [19].

6.1.2.1 Verification of visibility

Verification of visibility for tracking satellites

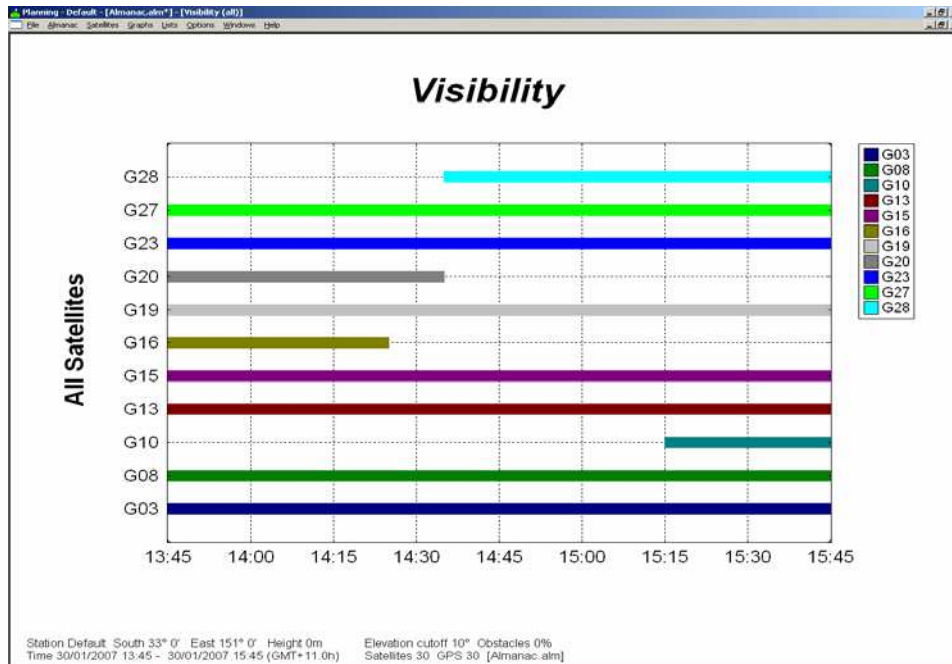


FIGURE 15. Screen capture of Visibility display of Planning software

6.1.2.2 Verification of elevation

Verification of Elevation for tracking satellites

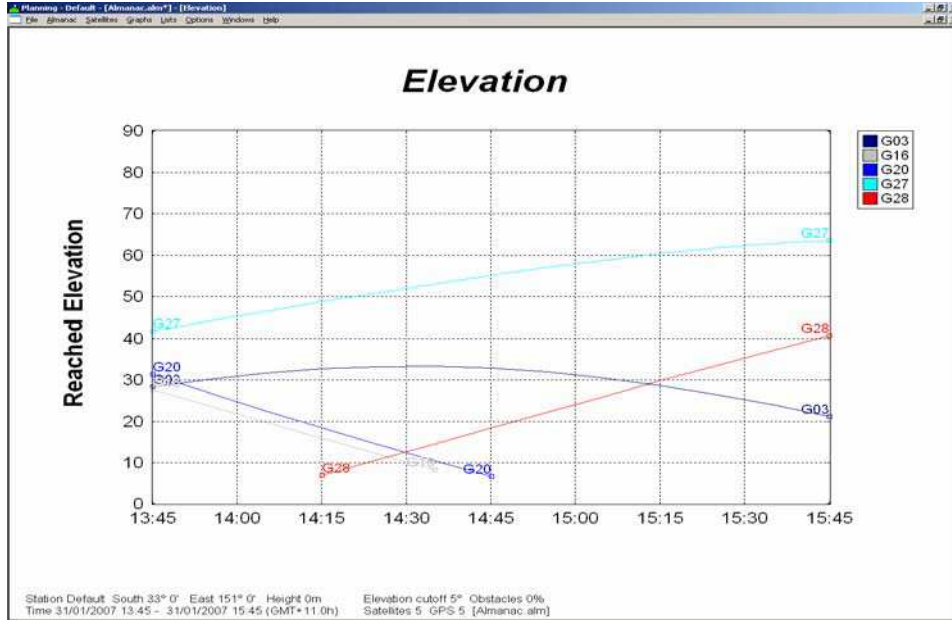


FIGURE 16. Screen capture of Elevation display of Planning software

6.1.2.3 Verification of azimuth

Verification of Azimuth for tracking satellites

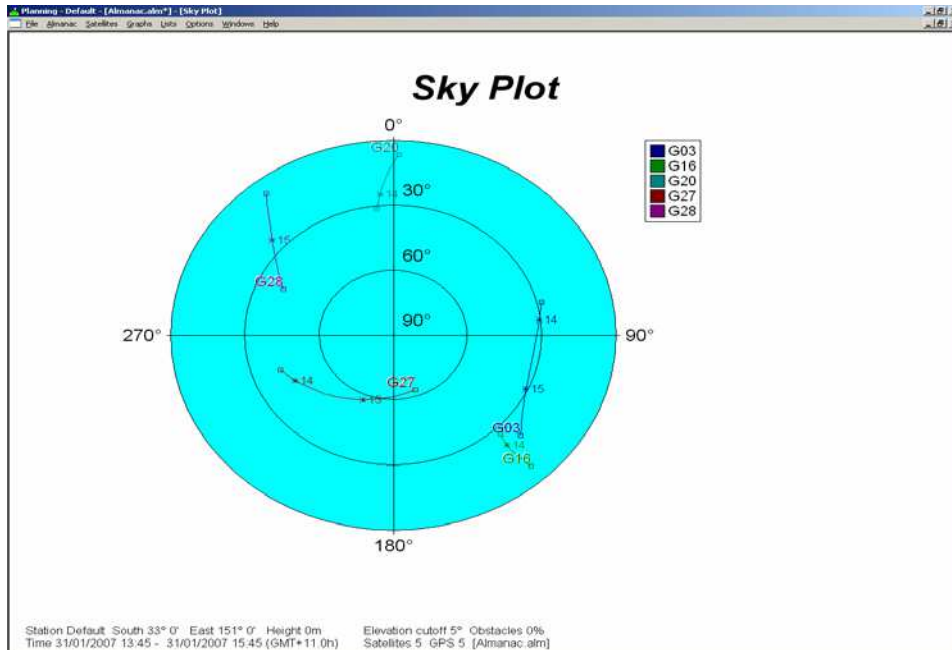


FIGURE 17. Screen capture of Sky Plot display of Planning software

6.2 Static Testing

After verifying if the receiver is tracking correct satellite with reliable information, X,Y,Z coordinate data was collected for a while by using DISPLAY_PLOT mode with 'ComTerm' that is discussed in chapter 5.5.4. The extracted X,Y,Z data was applied to MATLAB. The MATLAB script is available at **Appendix D**.

Based on the data, plots were generated for both the horizontal error and the height error. The plotting graph drawn is presented below.

The positioning solution is reasonably stable for the first 200 epochs, however after that, the software tends to drop satellites and the position solution fails. At around 500 epochs, instability in the solution can be seen. Considering that the atmosphere adjustment did not apply to this raw result yet, it provide reasonable position information with less than 30m errors for the height.

However there are two main problems can be observed from the given plotting graph. Firstly, the Namuru GNSS receiver tends to drop the satellites after it run for a while. Secondly, occasionally PVT information becomes incorrect. As a consequence, the information of pseudorange becomes unreliable.

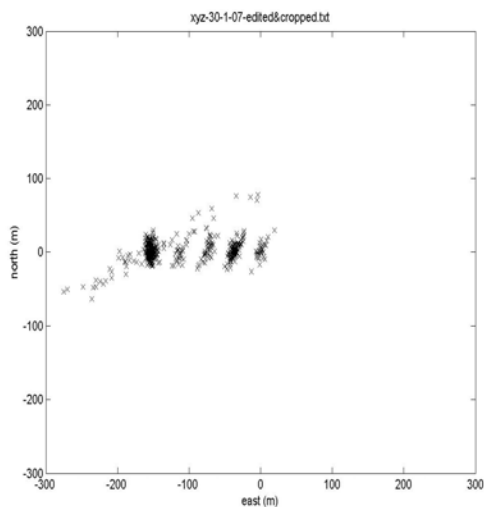


FIGURE 18. The horizontal error plots

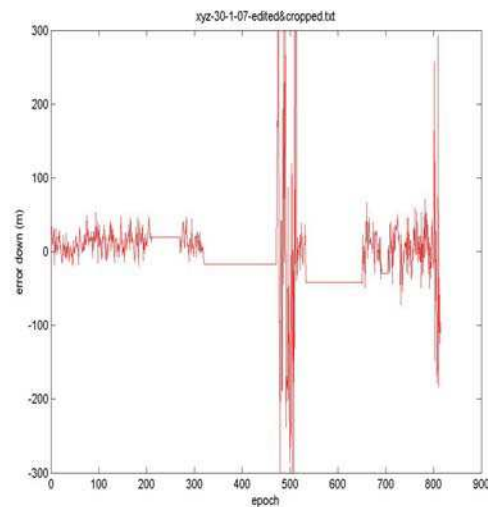


FIGURE 19. The horizontal error plots

Chapter 7 Conclusion and Future Work

7.1 Conclusion

Positioning data of X,Y,Z coordinates was collected. Based on the data, plotting graphs were drawn for both the horizontal error and the height error.

From the observation of result graph, the Namuru GNSS receiver has a capability of providing correct positioning information when it acquires more than four satellites. The PVT solutions are reasonably stable in the beginning. However, there are unsolved problems exist. After it runs for a while, occasionally it tends to drop satellites. As a result, the positioning information becomes unreliable. After the observation of unreliable behaviour of Namuru receiver, the possible problems were discussed. Tracking loops was suspected of this behaviour of the Namuru – GPL-GPS. On occasion there are very big clock corrections and the pseudoranges value becomes incorrect. As a result, the position solution also becomes incorrect. This may be caused by miss-locking onto one satellite signal and pulling the solution out.

In conclusion, the GPL-GPS software running on the Namuru GPS receiver has the capability to position correctly but more work is required to improve performance

7.2 Future work

The current Namuru GPS receiver with GPL-GPS software running on it is only the initial version of platform. And it can be enhanced and improved further in various ways in the future.

Future work can be summarized into three in general:

Firstly, the OSGPS project is still active [12]. And there is a significant progress in OSGPS. The latest version of OSGPS, which is version 2.0, contains features that GPL-GPS doesn't support such as:

- Atmospheric correction
- Carrier phase
- Better tracking loops
- Almanac load/save and processing via Flash

- Faster/better cold acquisition

Secondly, GPL-GPS is works however it is obvious that there are many bugs still exist inside of code. And below unsolved problem was pointed out.

- Poor tracking loops (2nd order PLL)
- Lacks atmospheric correction
- Poor acquisition and search on loss-of-signal
- Lacks carrier phase
- Almanac not stored in non-volatile memory
- eCos infrastructure needs more tuning
- Keep failing pull-in

Thirdly, there is a TODO file for GPL-GPS. This TODO file is attached in the **Appendix E**. And GPL-GPS is not complete GPS software rather it is an open source that is open to public for further development.

Bibliography

Journal Articles

1. Peter J Mumford, Kevin Parkinson, Andrew G Dempster,
The Namuru Open GNSS Research Receiver
2. Andrew Greenberg, Takuji Ebinuma,
Open Source Software For Commercial Off-the –Shelf GPS Receiver,
3. A G Dempster, K Parkinson, F Engel, P Mumford, C Rizos, G Heiser, The
UNSW/NICTA FPGA-based GPS receiver: A tool for GNSS research,
4. K J Parkinson, A G Dempster, P Mumford, C Rizos
'FPGA based GPS receiver design considerations'
The 2005 International Symposium on GNSS/GPS, Hong Kong
8-10 Dec, paper 8C-05
5. Namuru FPGA GPS Tracking module Data Sheet, Issue 1.2, Dec 2006
6. Andrew Greenberg,
OPEN SOURCE SOFTWARE FOR COMMERCIAL OFF-THE-SHELF GPS
RECEIVERS
Master degree Thesis Paper
May 6, 2005,
7. Clifford Kelly, Joel Barnes, Jingrong Cheng,
Open Source GPS Open Source Software for Learning about GPS, In 15th
international Tech. Meeting of the Satellite Division of the U.S.Inst. of
Navigation. Institute of Navigation, Sep 2002.
8. Engel F, Heiser G, Mumford P, Parkinson K, Rizos C(2004)
An Open GNSS Receiver Platform Architecture,
The 2004 International Symposium on GNSS/GPS, 6-8 Dec, 2004, Sydney.

9. K J Parkinson, A G Dempster, P Mumford, C Rizos,
Improving signal quality in FPGA based GPS receiver designs,
IGNSS (International Global Navigation Satellite Systems Society Symposium
2006 On GPS/GNSS , Surfers Paradise, Australia, 17-21 July

Web sites

10. Clifford kelley's OpenSourceGPS site
<http://home.earthlink.net/~cwkelley/>
11. Clifford kelley's OpenSourceGPS sourceforge site
<http://sourceforge.net/projects/osgps>
12. Takuji Ebinuma's ARMGPS site
<http://www.geocities.com/tebinuma/osgps/index.html>
13. Andrew Greenberg's GPL-GPS site
<http://gps.psas.pdx.edu/>
14. The "NAMURU" GPS project site
<http://dynamics.co.nz/gpsreceiver/>
15. Altera
<http://www.altera.com/>
16. SigNav Pty.Ltd
<http://www.signav.com.au/>
17. Trimble Planning
<http://www.trimble.com/>
18. The eCos real time operating system home page and on-line documentation
<http://ecos.sourceware.org/>

Appendix B Porting GPL-GPS to SigNavMG5001

STEP 1.

Download 'redboot.zip' and extract it.

Connect UART1 of the SigNav MG5001 receiver to a serial port of host computer

Connect power cable to the board and set the boot pin to 'Load'

Open HyperTerminal, and start 'New connection' with below set up

Connect using 'COM1'

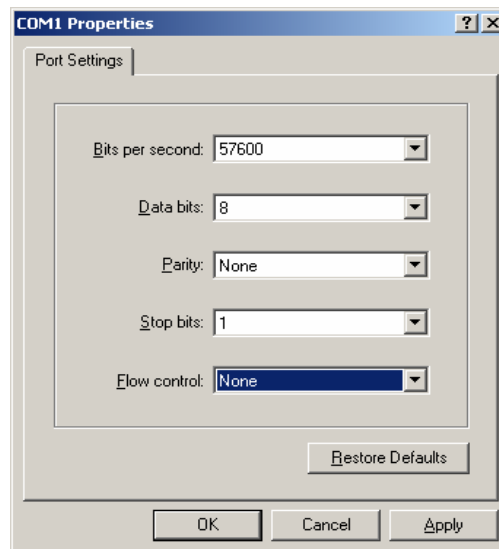
Bits per second: '57600'

Data bits : 8

Parity : None

Stop bits : 1

Flow control : None



STEP 2.

Open the connection to HyperTerminal by clicking 'Call' from the menu

Choose 'Transfer' => 'Send Text File...' from the menu

Send 'ram_download.bin' which is generated by extracting 'redboot.zip'

The LED7 (GPIO7) is flashing during download.

After finishing download, you should see the RedBoot welcome messages.

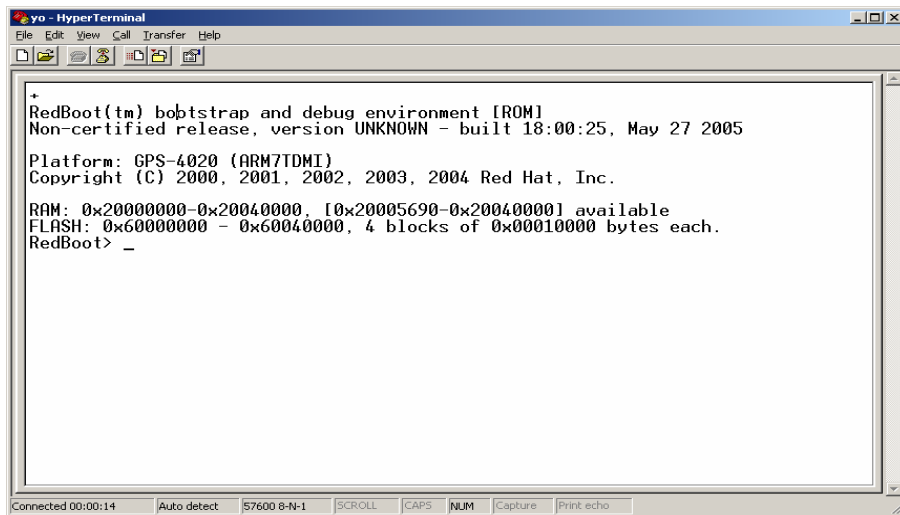
STEP 3.

Then burn redboot into flash

```
RedBoot> fis write -f 0x60000000 -b -0x20020000 -l 0x10000
```

STEP 4.

After burn the redboot into flash, Clicking switch at MG5001 you get below message following command line prompt

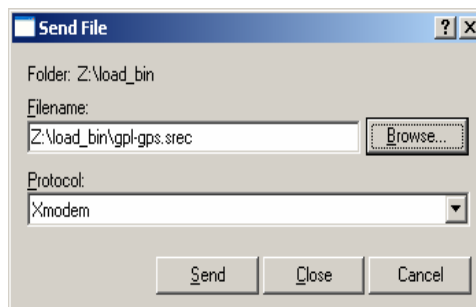


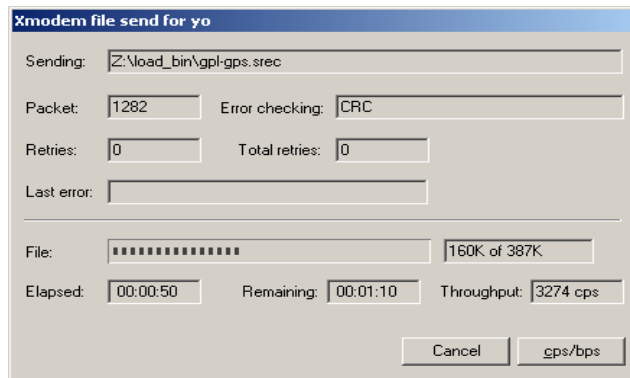
STEP 5.

Then type 'load -v -m xmodem' in RedBoot command line then enter

```
RedBoot> load -v -m xmodem
```

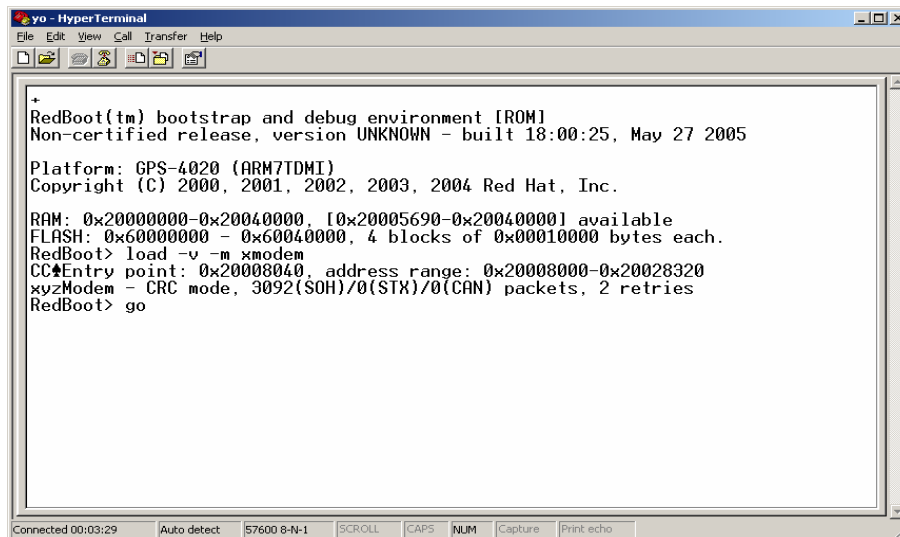
And 'Transfer'=> 'Send file' from the top menu of HyperTerminal and enter Set up the protocol as Xmodem then Browse the file 'gpl-gps.srec'





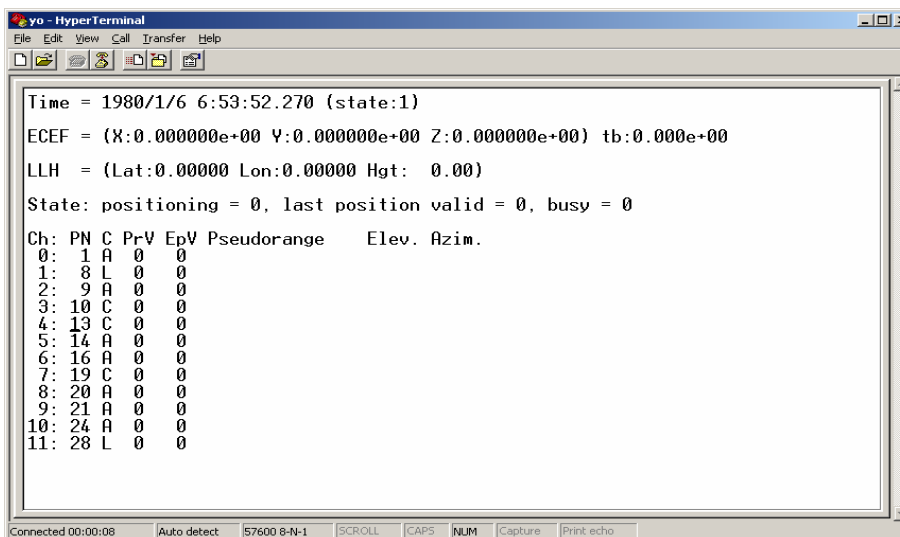
type 'go' on HyperTerminal and enter to execute the software.

RedBoot> go



Then switch the serial cable to Port 2 because the message has been sent out to Port2.

The below screen capture will be displayed on HyperTerminal.



Appendix C Makefile for GPL-GPS

```
# Makefile for 'NAMURU OpenSource GPS receiver'
# University of New South Wales
# School of Surveying and Spatial Information Systems
# Open Source Namuru GPS receiver project
# date : 21st/Dec/2006

ECOS_DIR = $(INSTALL_DIR)

# INSTALL_DIR=d:/namuru/ecos_a/ecos_a_install
include $(ECOS_DIR)/include/pkgconf/ecos.mak

RM = rm -f
CC = $(ECOS_COMMAND_PREFIX)gcc
LD = $(CC)

# ECOS_GLOBAL_CFLAGS are defined in $(ECOS_DIR)/include/pkgconf/ecos.mak

CFLAGS      = -I$(INSTALL_DIR)/include
CXXFLAGS    = $(CFLAGS)
LDFLAGS     = -nostartfiles -L$(INSTALL_DIR)/lib -Ttarget.ld
OBJS        = gpl-gps.o allocate.o display.o ephemeris.o interrupts.o \
              log.o measure.o message.o position.o pseudorange.o serial.o time.o tracking.o \
              namuru_io.o namuru_io.o namuru_corraddr.o led.o # added file by yong

.PHONY: all clean

all: namuru

clean:
    -$(RM) namuru $(OBJS) # $(DEF) yong

tags:
```

```

ctags *.ch]

SRC = $(wildcard *.c)

# Rules
# Handle dependencies automatically in (GNU-make/gcc)-style
%.d: %.c
    $(CC) -isystem $(ECOS_DIR)/include -MM $(CPPFLAGS) $< > $@.$$$$; \
    sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
    rm -f $@.$$$$

# Include auto-dependency files
include $(SRC:.c=.d)

namuru: $(SRC:.c=.o)
    $(LD) $(LDFLAGS) $(ECOS_GLOBAL_LDFLAGS) $(SRC:.c=.o) -o $@

```


Appendix D MATLAB script

```
% MATLAB script that used to generate the plotting graph,  
% it calls some functions from a toolbox  
% function to plot trajectory in ned coordinate system  
% written by Peter Mumford 5/July/06  
  
function trajectory(filename);  
  
% Input file name to be plotted.  
fid = fopen(filename,'r'); % open file  
if(fid == -1) display('error on file read...');  
end  
  
[vec,count] = fscanf(fid,'%f%f%f%f',[4,inf]);  
  
% note that %f works better than using %i to pick up the integer count;  
vec_len = length(vec)  
  
x = vec(1,:);  
y = vec(2,:);  
z = vec(3,:);  
off = vec(4,:);  
%drift = vec(5,:);  
  
% north pillar  
  
centerXYZ = [-4644468.695 2549957.976 -3538921.139]; %GDA94  
  
[center_lat,center_lon,h] =  
cart_to_geo_WGS84(centerXYZ(1),centerXYZ(2),centerXYZ(3));  
  
dx = (centerXYZ(1) - x)';  
dy = (centerXYZ(2) - y)';
```

```

dz = (centerXYZ(3) - z)';

% convert to north, east, down coordinate system
[ned] = ecef2ned([dx,dy,dz],[center_lat,center_lon,h]);

% plot
fig1 = figure;

NE_plot = plot(ned(:,2),ned(:,1),'kx'); % kx for black cross

set(gca,'YLim',[-300 300],'XLim',[-300 300]);
ylabel('north (m)');
xlabel('east (m)');
set(gcf,'Name','namuru-gpl');
title(filename);

% plot
fig2 = figure;
D_plot = plot(ned(:,3),'r');

%set(gca,'YLim',[-100 100]);
xlabel('epoch');
ylabel('error down (m)');
set(gcf,'Name','down-errors');
title(filename);
set(gca,'YLim',[-300 300]);

mean_down_error = mean(ned(:,3))

% close files
if(fid ~= -1)
    fclose(fid);
end

```

Appendix E TODO file for GPL-GPS

1. Check the stack usage on the idle thread - can we reduce it to the minimum 1200 bytes? The 1200 bytes is defined as the minimal stack size for us -- sorry I forgot where. Also maybe we should change that since we clearly have threads that take less than 512 bytes (see the 'd' option in the display).
(Isn't stack usage defined in gpl-gps.c ?)
2. Check the tm_basic code to find out how to measure the interrupt stack usage.
3. In the config tool, under Math Library -> Compatibility Mode, maybe we can use IEEE - only? But need to be careful if we ever need POSIX support.
4. Move floating point variables in position.c to statics, so that we can reduce the stack down to a normal thread size. Seems like it's be more efficient, in terms of not having to have a huge stack (and being more deterministic about stack stuff). (Check efficiency too.)
5. Take the long term average of the clock correction, and add it to the clock every TIC if we're not positioning. (If we're not positioning?)
6. In position.c, change to a structure indexed ONLY by channel. Probably means having another array to index the valid channels? Or?
7. Implement a "wider" aquisition algorithm, possibly DFT based.
8. Modularize the algorithms to allow substitution of various forms.
9. Change the navigation bit extraction in tracking.c to be noise-tolerant. It is desirable to be able to choose either the present extraction or the proposed one, see Modularization above.
10. Implement (re-implement?) carrier phase aiding

11. Implement a Kalman filter
12. Dynamic loading of code?
13. A command interpreter?
14. Provide an option for fixed point computations
15. High-bandwidth communications with the host for debugging or off-loading some of the computation.
16. Achieve some host independence, so that algorithms can run on a PC for example.