Design Document for G2 Airborne SAR Processor

J.M. Horrell

File: filename
Document No: rrsg:00

Contents

List of Figures

List of Tables

Chapter 1

Scope

1.1 Identification

This is the design document for the G2 airborne SAR processor.

1.2 Purpose

The document describes in detail the design of the G2 processor. The intention is to provide a clear map with interface definitions to assist in further development of the processor. This document contains the technical detail required by developers and is not intended as a users manual which exists as a separate document.

1.3 Introduction

The G2 processor has is based on the range-Doppler algorithm and has been designed for the processing of airborne synthetic aperture radar data. The processor has been written to support the flexible nature of experimental work and thus is highly configurable with a modular design. The core modules[1] are written in C for speed with integration of the various modules provided through Python code.

The implementation described here using Python modules as the overall "glue" has been directed at the processing of SASAR VHF data (the South African Synthetic Aperture Radar VHF Sensor) and is currently the official (and only) processor for this radar. However, many of the modules may be reused for processing data from other sensors. For example, in addition to SASAR VHF data, the core modules of the processor have on a number of occasions been used to process E-SAR data from the DLR, Germany and also to process simulated SAR data using Rolf's Radar Simulator of the UCT RRSG.

In this document, the functional requirements for the code are stated and the architectual- and detailed designs are presented.

[1]The major core modules for range compression, "rngcom", and azimuth compression, "azcom", have been the result of an evolutionary development rather than a top down design and, while fully functional and commented, could be rewritten with more internal modular structure to allow for easier future development.

Chapter 2

Requirements

The G2 processor has been developed as an in-house capability of the Radar Remote Sensing Group and thus has not been the result of a set of customer requirements.

The in-house requirements have been to develop a flexible SAR processor for experimental work which incorporates the full functionality required for airborne SAR processing of real systems (i.e includes motion compensation, checks for I/Q imbalance, multilook processing).

Early in the development, computer RAM was limited and the processor was required to operate with a configurable amount of memory. In addition, the processor was required to be fast in execution. The memory and execution speed requirements have been largely maintained up to the latest version, but over time, there has been some tradeoff introduced between these requirements and maintainability.

The processor was required to be able to handle a large range of sensor configurations from medium resolution VHF (100 MHz+) data to high resolution at higher frequencies. The processor limitations at low frequencies are detailed in [1].

The requirement was for the code to run on Unix-like systems, but portability issues have been considered at all stages of development. The current development and run-time OS is predominantly Linux. Early versions of the processor ran under DOS and Solaris and the latest integrated version should be able to be fairly easily ported to the win32 platforms.

Chapter 3

Architectural Design

3.1 Overview

The G2 sythetic aperture radar (SAR) processor is based on the range-Doppler algorithm and has been designed for the processing of airborne SAR data. The processor is modular and flexible and can handle a wide range of SAR processing tasks. This version of the processor includes a Python program, "g2.py", which integrates the various modules and has been designed for the processing of SASAR VHF data (South African SAR VHF sensor). The integrated processor runs from the command line and reads its configuration parameters from the ASCII processor configuration file.

In many cases, it is possible to use the integrated version for processing of data from systems other than the SASAR VHF system by simply changing input parameters. Where the integrated processor is found not to be suitable for a particular processing job, it might still be possible to configure the various modules manually to get the job done (see below). In addition, if semi-automated processing is required for a new SAR sensor, a new version of the overall glue program in Python could be created which used the same core modules (such as range compression, azimuth compression, etc.).

3.2 Design Philosophy

The processor has been written to be as flexible as possible (required for experimental systems) whilst retaining a simple user interface. The processor is run from the command line with setup parameters specified in an ASCII configuration file.

The integrated processor is comprised of a number of modules, each of which usually reads from some input file and writes to an output file. Log files are also generated by certain of the modules. These are read by later modules to set parameters in the integrated processor. For usability, all module configuration files are ASCII and, in order to save disk space, most large data files are binary.

The operator can, through the processor configuration file, decide whether to keep all the intermediate temporary files and temporary log files or delete them at the end. For debugging a processing run, it is often very useful having all the temporary log- and config (and even data) files available, but these can take up a lot of disk space.

Most of the modules are written in C (for speed) with certain modules in the Python language. The overall glue for the processor is written in Python. The C modules have been compiled to be standalone executables and the Python modules to Python byte code which are called by the main Python code. The Python code automatically configures the processing for the particular modules comprising the integrated processor. Thus, the user is only required to set up one ASCII file which specfies the overall configuration for the processing run. Before each module is executed, the specific configuration file for that module is automatically configured.

For example, the azimuth processing module requires a configuration ASCII text file which the Python code automatically generates before calling the operating system to excute azimuth processing.

While using the integrated processor is the best way for processing SASAR VHF (and similar) data, an alternative approach is available in that the separate modules comprising the integrated processor may be run independently from the command line. This is a deliberate part of the design to allow for greater flexibility in usage.

3.3 Code Structure

3.3.1 Main Program

The top-level (main) program for the G2 processor is named "g2.py" and is controlled through the command line. This Python program provides the glue which ties together all the various modules.

Apart from calling the relevant processing modules, g2.py along with its helper modules g2tmpl.py and g2tools.py, are also responsible for all the logic required to string the various processing modules together, the auto-generation of template files (including the G2 User Manual in html), on-the-fly generation of the various processing module configuration files, error message handling, and handling of certain other functionality (like the clean up).

g2.py relies heavily on two configuration files being properly set up before a processing run. These are the processor config file and the radar config file. These are both ASCII text files for which commented templates may be optionally generated by g2.py.

1) processor configuration file - this is configured at process time by the processor operator and contains all those parameters which the processor operator usually would select (like the required azimuth resolution). For normal operation, the processor config file would be the ONLY file with which the processor operator needs to be concerned.

2) radar configuration file - this ideally written by the radar control software and would not normally be altered by the processor operator. However, even for the current SASAR system, this file is required to be set up manually. Parameters in the radar configuration file include radar centre frequency and pulse length, for example. The radar config file also contains names (without paths) of the raw data file, LBR file, DGPS file, etc. The paths to those files are set up in the processor config file.

3.3.2 Processing Modules

In the main integrated processor configuration file, processing modules may be switched in and out. Permissible states for modules are are y / n / only /off which allows for very flexible operation. The meaning of these states are:

- **y** (yes) - Run this module, performing relevant parameter calculations or reads from log files, and write log file, if any.
- **n** (no) - Do not run this module, but infer parameters and read from log files as if it had been run.
- **only** - Run only this module and none other. Dynamic parameters are calculated or read from intermediate log files of the other modules unless these are set to "off"
- **off** - Do not run this module and do not infer any parameters or read from its log file (i.e. as if module absent altogrether). Note that the 'off' state is only permitted for certain modules.

Checks are performed for modules not in 'off' state which depend on a module which is in the 'off state. These options allow flexible processor configuration. For example, a run may be restarted from half-way, by setting the previously completed portions to "n". The correct parameters for the later modules will still be calculated or read from the intermediate log files.

The main processing modules (available as standalone modules, unless contra-indicated) which may be configured in the processor configuration file are (in normal order of SASAR processing execution):

- **UnpackIMU** - Unpack the IMU records from the SASAR LBR file and write to an ASCII file (C executable - “imu_unpack”).
- **UnpackDGPS** -Unpack the DGPS records to a more readable ASCII format and sync with the IMU data (Python code - “g2unpk_dgps.py”).
- **MergeMocData** - Merge the IMU and DGPS records to form a single ASCII file with the LBR PRI, Latitude, Longitude, etc. This does all the smoothing, interpolation, etc. (Python code “g2mocfilt.py”). This also requires the Scientific Python modules be installed as freely available on the web.
- **MocompCalc** - Calculate the range shifts required for each range line from the merged motion data. Also creates the geocoding information. (C executable - “mocomp”).
- **PlotMotionError** - Plot the motion compensation range shift as calculated by the MocompCalc module. (Python code - not standalone). This requires that the freely available Gnuplot.py module be installed.
- **SniffDC** - Calculate the DC offsets and average I to Q value ratio from an analysis of part of the raw data (C executable - “sniffdc”)
- **RngProc** - Range compression, interference suppression and motion compensation correction implementation (C executable - “rngcom”).
- **StepFreqProc** - Step frequency processing (C executable - “stepf”).
- **CornerTurn** - Corner turn the range compressed file (C executable - “corner”).

- **AzProc** - Range curvature correction, azimuth compression and multilook (C executable - “azcom”).
- **Float2Tiff** - Convert floating point output from azimuth compression to TIFF file for easy viewing (various C executables, all working on binary data - “iq2mag”, “flt2byte”, “b2tif”). “iq2mag” converts complex data to magnitude or power, “flt2byte” converts floating point data to unsigned char, “b2tif” creates TIFF format file from unsigned char data.
- **EndianSwap -** Swap endian format (C executable - “swapend”).
- **Orient** - Convert from azimuth line format to range line format (uses corner turn C executable - “corner”)
- **ImageLog** - Create image log file with geocoding info, etc. (Python code - not standalone)
- **CleanUp** - Remove all temporary data and log files (Python code - not standalone).

Chapter 4

Detailed Design

Note: the built-in Python and C header files and functions are not included in the sections below.

4.1 G2 Processor Main Program - g2.py

Name:

Python program in file g2.py.

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
configuration_file	ASCII file	in	contains the processor config parameters
image_log_file	ASCII file	out	log of processed image parameters
image_data_file	binary float	out	4-byte floating point binary output
image_tiff_file	binary	out	TIFF format image file

Functionality:

Reads in the processor- and radar configuration parameters, sets up configuration files for the various processing modules on-the-fly, performs error handling, cleans up temporary files.

Dependancies/Resources used:

Small memory footprint.

Requires files:

- Gnuplot.py (Python interface to gnuplot plotting program)
- g2tmpl.py (g2tmpl module, contains functions only)
- g2tools.py (g2tools module, defines class G2Config)

Subordinates:

The following functions from the g2tmpl module are called:

- printHelp()
- createRadarCfgTmpl()
- createProcCfgTmpl()
- createHTMLUserManual()

The following functions from the g2tools module are called:

- G2Config()
- addCfgFile()
- versionControl()
- checkForOnly()
- inferParams()
- initChecks()
- parseParam()
- createStepFreqUserFile()
- createRngProcCmdFile()
- createStepFreqProcCmdFile()
- createAzProcCmdFile()
- writeImageLog()

The following standalone executables are called:

- imu_unpack
- g2unpk_dgps.py
- g2mocfilt.py
- mocomp
- sniffdc
- corner
- iq2mag
- flt2byte
- b2tif
- swapend

4.2 g2tmpl - createRadarCfgFile()

Name:

createRadarCfgFile() in file g2tmpl.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
filename	string	in	file to which config template written
<return value>	int	out	0 = success

Functionality:

Writes out a template radar configuration file including notes on usage.

Dependancies/Resources used:

Small memory footprint.

Subordinates:

none

4.3 g2tmpl - createProcCfgFile()

Name:

createProcCfgFile() in file g2tmpl.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
filename	string	in	file to which config template written
<return value>	int	out	0 = success

Functionality:

Writes out a template processor configuration file including notes on usage.

Dependancies/Resources used:

Small memory footprint.

Subordinates:

none

4.4 g2tmpl - printHelp()

Name:

printHelp() in file g2tmpl.py

Interface:

N/A

Functionality:

Print quick help information to screen

Dependancies/Resources used

N/A

Subordinates:

none

4.5 g2tmpl - createHTMLUserManual()

Name:

createHTMLUserManual() in file g2tmpl.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
filename	string	in	file name to which to write User Manual
<return value>	int	out	0 = success

Functionality:

Writes out the G2 User Manual in HTML format.

Dependancies/Resources used:

N/A

Subordinates:

none

4.6 g2tools - addCfgFile()

Name:

class G2Config, method addCfgFile() in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
FileName	string	in	input configuration file name
<return value>	int	out	0 = success

Functionality:

Add the contents of the configuration file to the current object's 'cfg' dictionary.

Dependancies/Resources used:

N/A

Subordinates:

none

4.7 g2tools - parseParam()

Name:

class G2Config, method parseParam() in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
FileName	string	in	name of file to search
SearchStr	string	in	string to find
KeyStr	string	in	dictionary key to use for adding parameter
dict	dictionary	in	name of dictionary to use
<return value>	int	out	0 = success, -1 = error

Functionality:

Parses a specified ASCII file reading each line, and looks for a specified substring in the line read. If located, finds the colon and then adds the parameter value specified after the colon to a specified Python dictionary object (either "cfg" or "wrk").

Dependancies/Resources used:

N/A

Subordinates:

none

4.8 g2tools - convSecsToTimeStr()

Name:

class G2Config, method conSecsToTimeStr(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
inSecs	float	in	number of seconds
KeyStr	string	in	dictionary key for result
<return value>	int	out	0 = success

Functionality:

Convert time in seconds to hours, minutes and seconds.

Dependancies/Resources used:

N/A/

Subordinates:

none

4.9 g2tools - convDegToDegMinSec()

Name:

class G2Config, method conDegToDegMinSec(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
inDeg	float	in	number of degrees
KeyStr	string	in	dictionary key for result
<return value>	int	out	0 = success

Functionality:

Convert angle in degrees to degrees, minutes and seconds.

Dependancies/Resources used:

N/A

Subordinates:

none

4.10 g2tools - displayCfgContents()

Name:

class G2Config, method displayCfgContents in file g2tools.py

Interface:

None

Functionality:

Display the contents of the “cfg” dictionary.

Dependancies/Resources used:

N/A

Subordinates:

none

4.11 g2tools - displayWrkContents()

Name:

class G2Config, method displayWrkContents in file g2tools.py

Interface:

None

Functionality:

Display the contents of the “wrk” dictionary.

Dependancies/Resources used:

N/A

Subordinates:

none

4.12 g2tools - versionControl()

Name:

class G2Config, method versionControl in file g2tools.py

Interface:

None

Functionality:

Enable backwards compatibility with previous versions of the radar configuration file and the processor configuration file.

Dependancies/Resources used:

N/A

Subordinates:

none

4.13 g2tools - disableAll()

Name:

class G2Config, method disableAll() in file g2tools.py

Interface:

N/A

Functionality:

Switch all processing modules to “n” if they are not already flagged as “off”.

Dependancies/Resources used:

N/A

Subordinates:

none

4.14 g2tools - checkForOnly()

Name:

class G2Config, method checkForOnly() in file g2tools.py

Interface:

N/A

Functionality:

Check through all the processing module flags to find first one (if any) flagged as “only”. If found, disable all modules by calling the disableAll() method and then set the module flagged as “only” to “y”.

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn’t adequately describe operatiuon>

Dependancies/Resources used:

N/A

Subordinates:

- disableAll()

4.15 g2tools - initChecks()

Name:

class G2Config, method initChecks(), in file g2tools.py

Interface:

N/A

Functionality:

Perform some consistency and logic checks at start of processing run.

Dependancies/Resources used:

N/A

Subordinates:

none

4.16 g2tools - inferParams()

Name:

class G2Config, method inferParams(), in file g2tools.py

Interface:

N/A

Functionality:

Infer processing parameters based on the configuration parameters supplied.

Dependancies/Resources used:

N/A

Subordinates:

none

4.17 g2tools - createStepFreqUserFile()

Name:

class G2Config, method createStepFreqUserFile(), in file g2tools.py

Interface:

N/A

Functionality:

Create a step frequency processing user file. This is required by rngcom for normal step freq mode. Uses parameters in the "cfg" and "wrk" dictionaries.

Dependancies/Resources used:

N/A

Subordinates:

none

4.18 g2tools - calcAzProcParams()

Name:

class G2Config, method calcAzProcParams(), in file g2tools.py

Interface:

N/A

Functionality:

Set up some parameters for azimuth processing. Uses parameters in the “cfg” dictionary.

Dependancies/Resources used:

N/A

Subordinates:

none

4.19 g2tools - writeImageLog()

Name:

class G2Config, method writeImageLog(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
ILogName	string	in	image log file name
MLogName	string	in	motion log file name
RLogName	string	in	range processing log file name
ALogName	string	in	azimuth processing log file name
<return value>	int	out	0 = success

Functionality:

Parse the various log files from the different processing modules and write out the image log file. The image log file contains mostly image specific information.

Dependancies/Resources used:

N/A

Subordinates:

- parseParam()
- convSecsToTimeStr()
- convDegToDegMinSec()

4.20 g2tools - createRngProcCmdFile()

Name:

class G2Config, method createRngProcCmdFile(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
FileName	string	in	name for range processing config file
<return value>	int	out	0 = success, -1 = error

Functionality:

Creates the range processing (rngcom) configuration file using parameters from the "cfg" and "wrk" dictionaries.

Dependancies/Resources used:

N/A

Subordinates:

none

4.21 g2tools - createStepFreqProcCmdFile()

Name:

class G2Config, method createStepFreqProcCmdFile(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
FileName	string	in	name for step freq processing config file
<return value>	int	out	0 = success, -1 = error

Functionality:

Creates the step frequency processing (stepf) configuration file using parameters from the "cfg" and "wrk" dictionaries.

Dependancies/Resources used:

N/A

Subordinates:

none

4.22 g2tools - createAzProcCmdFile()

Name:

class G2Config, method createAzProcCmdFile(), in file g2tools.py

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
FileName	string	in	name for azimuth processing config file
<return value>	int	out	0 = success, -1 = error

Functionality:

Creates the azimuth processing (azcom) configuration file using parameters from the “cfg” and “wrk” dictionaries.

Dependancies/Resources used:

N/A

Subordinates:

none

4.23 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.24 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.25 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.26 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.27 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.28 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.29 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.30 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.31 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.32 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.33 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.34 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.35 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.36 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

4.37 <module name>

Name:

<module/function name>, in file <filename.c>

Interface:

The following table describes the API for this routine:

parameter	type	in/out	description
<return value>	int	out	0 = success, -1 = error

Functionality:

<describe the operation/functionality of this function>

Pseudo-code:

<optional: include pseudo-code or flowchart here if above section doesn't adequately describe operatiuon>

Dependancies/Resources used:

<describe memory and any other system resources used>

In addition, the following header files are required:

- <list of header files>

Subordinates:

<functions called from this one>

Library:

This routine is contained in <library-name>

Bibliography

[1] J.M. Horrell, “Range-Doppler Synthetic Aperture Radar Processing at VHF Frequencies”, Ph.D. Thesis, University of Cape Town, May 1999.