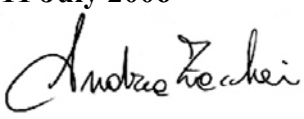
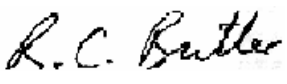
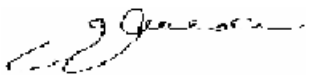
TITLE: # Planck LFI – Onboard Processing and Ground Segment Simulator Software

DOC. TYPE: TECHNICAL NOTE

PROJECT REF.: PL-LFI-OAT-TN–035 **PAGE:** I of IV, 09

ISSUE/REV.: 1.0 **DATE:** 11 July 2006

Issued by	**Michele Maris** **Marco Frailis**	**Date:** **Signature:**	**11 July 2006** 
Agreed by	**A. ZACCHEI** **LFI DPC Manager**	**Date:** **Signature:**	**11 July 2006** 
Approved by	**R.C. BUTLER** **LFI Program Manager**	**Date:** **Signature:**	**11 July 2006** 
Approved by	**N. MANDOLESI** **LFI Principal Investigator**	**Date:** **Signature:**	**11 July 2006** 

DISTRIBUTION LIST

Recipient	Company / Institute	E-mail address	Sent
A. Zacchei	INAF-OATs	zacchei@ts.astro.it	
M. Maris	INAF-OATs	maris@ts.astro.it	
M. Frailis	INAF-OATs	frailis@ts.astro.it	

CHANGE RECORD

Issue	Date	Sheet	Description of Change	Release
0	2006 July 11		Starting of the document	0
1	2006 July 11		First Issue	0

TABLE OF CONTENTS

1 SCOPE

This document describes the module simulating the onboard processing of PType 2 data and the subsequent decoding and conversion to TOI in the DPC L1. This module is a partial answer to the content of [RD-1].

The scope of the module is to allow the estimation of processing errors for given processing parameters (see [AD-1 and RD-2] for the list of processing parameters, the processing algorithm and the exact definition of PTypes), to simulate the effect of processing errors on real data streams.

The module is designed to work both stand-alone or as part of a more complex pipeline.

1.1 LIMITS OF APPLICABILITY

The module just simulates the PType 2 on-board processing and the subsequent decoding.

The module does not support any other kinds of processing.

The module does not simulate any other intermediate steps of processing (compression/decompression, packeting and unpacking of scientific tm) being expected to be transparent to the user.

The module does not simulate the ADC acquisition.

The module is designed to work under LINUX/UNIX.

The module is NOT DESIGNED to work under MS-DOS or any other operative systems different from LINUX.

The authors WILL NOT PROVIDE any support or any explanation to any attempt to port this code on any other operative system.

2 APPLICABLE/REFERENCE DOCUMENTS

2.1 APPLICABLE DOCUMENTS

[AD-1] Planck-LFI Communications, ICD,
M. Miccolis
PL-LFI-PST-ID –013, Version 3.0, January 2004

2.2 REFERENCE DOCUMENTS

[RD-1] Prospects in simulating onboard data acquisition for LFI within the LS – L1 pipeline.
M. Maris
PL-LFI-OAT-ME-015, 21 Feb 2005

[RD-2] Reconfiguration for LFI on-board data processing and scientific telemetry
M. Miccolis, A. Mennella, M. Bersanelli, M. Maris
PL-LFI-PST-TN-037, Issue 1.0, March 2003

2.3 ACRONYMS LIST

ADU	Analog / Digital Unit
FM	Flight Model
FP	Floating Point
PType	Processing Type
QM	Qualification Model
TMH	Telemetry Handler
TMU	Telemetry Unscrambler
TQL	Telemetry Quick-Look
OBT	On Board Time
TOI	Time Ordered Information
TOD	Time Ordered Data

3 OPERATIONS

This code simulates part of the onboard processing of Planck / LFI, in particular it simulates the activity of PType 2 processing from PType 1 data.

It also simulates Ground Segment processing from PType 2 to TOIs and computes the processing error (quantization noise).

The code accepts in intput TOIs of PType 1 data generated by the TMH/TQL or simulated TODs generated by the LS, the format of input files is in sect. 4.6.

The code applies a given set of REBA parameters to the input file (the same set to all the data) and generates output files containing processing data and the processing noise.

The code operates through the standard command-line parameters passing interface.

Read carefully the Warning in Assignin Naver.

4 THE CODE

The code is written in C++. It is divided into a C++ library file and a C++ main file.

In addition, to test the code, a C++ generator of FITS input files is provided.

The list of files is the following

`quantum.cc`	Library File
`quantum_main.cc`	the main program
`quantum_gen_example.cc`	the example generation program

4.1 INSTALLATION AND COMPILATION

4.1.1 REQUIREMENTS

g++ 3.3.1 or higher.
cfitisio v.3.0

4.1.2 INSTALLATION

Copy the tar.gz file in the directory where the installation it is wanted and unpack it.

4.1.3 COMPILATION

To compile the simulation code

```
g++ -o lfi_obc_sym.x quantum_main.cc -L /usr/local/lib/ -lm -lcfitsio
```

To compile the file example generation code

```
g++ -o quantum_gen_example.x quantum_gen_example.cc -L /usr/local/lib/ -lm -
lcfitsio
```

An example of compilation is in the file `example.sh`.

4.1.4 EXECUTION

An example of execution is in the file `example.sh`.

The command

```
./lfi_obc_sym.x
```

shows an help.

The command

```
./lfi_obc_sym.x file_test1 b example_scale.fits 1 0. 1. 1. 0
```

runs the code taking in input the file “example_scale.fits” assuming Naver = 1, offset_adjust=0., second_quant = 1., gmf1=1., gmf2=0. and generating output files with names with prefix “file_test1”. The binary is in “b” format (big endian). The input file is assumed to be in the current directory the output is written in the current directory.

4.2 THE LIBRARY

The library file `quantum.cc` contains three C++ routines performing the PType 2 processing, the PType 2 decoding and the evaluation of the quantization errors.

All the functions are of int type giving in output a integer 0 code if the operations runs properly.

REBA parameters:
float: second_quant, offset_adjust, gmf1, gmf2
integer: naver

Arrays
`long nelements` = the number of elements in the arrays
`sky, load` = float arrays of sky and load signal
`q1, q2` = integer arrays of mixed and re-quantized data
`rsky, rload` = float arrays of reconstructed sky and load signal
`esky, eload` = float arrays of sky and load processing errors.

All the arrays are assumed to be of equal length.

The functions do not allocate arrays which are assumed to be already allocated by the calling program.

4.2.1 LFI_OBC

```
int lfi_obc(int *q1, int *q2, float *sky, float *load, long
nelements, float second_quant, float offset_adjust, float gmf1,
float gmf2, int naver)
```

Performs the transformation from sky, load signals to q1, q2 according to [RD-1].

4.2.2 LFI_GS

```
int lfi_gs(float *sky, float *load, int *q1, int *q2, long
nelements, float second_quant, float offset_adjust, float gmf1,
float gmf2, int naver)
```

Performs the transformation from q1, q2 to sky, load reversing the action of **lfi_obc**.

4.2.3 LFI_PROC_ERR

```
int lfi_proc_err(float *esky, float *eload, float *rsky, float
*rload, float *sky, float *load, long nelements, float
second_quant, float offset_adjust, float gmf1, float gmf2, int
naver)
```

Computes the processing error defined as:

esky = rsky – sky,

eload = rload – load.

4.3 THE MAIN CODE

The main code manages the handling of input, output files and of REBA parameter, handles the allocation of memory and performs the related computations.

The parameters are passed through the command line as follow

```
./lfi_obc_sym.x <<output-file>> <<output-type>>  <<input-file>>
<<naver>> <<offsett>> <<quant>> <<gmf1>> <<gmf2>>
```

where

<<output-file>>	prefix for the name of the output file (without extension)
<<output-type>>	'b' for binary in big endian 'l' in little endian
<<input-file>>	name of input file
<<naver>>	naver parameter (int)
<<offset>>	offset (float)
<<quant>>	quantization factor (float)
<<gmf1>>	gmf1 (float)
<<gmf2>>	gmf2 (float)

The input file has to be a FITS file formatted as explained in sect. 4.6.

The code will generate in output four files named as follow

<<output-file>>_q.fits	q1, q2 arrays, FITS file
<<output-file>>_r.fits	rsky, rload arrays, FITS file
<<output-file>>_e.fits	esky, eload arrays, FITS file
<<output-file>>.bin	q1, q2 in little-endian, interlaced

As an example if <<output-file>> := `test1` then the following files will be generated: `test1_q.fits`, `test1_r.fits`, `test1_e.fits`, `test1.bin`.

4.4 THE EXAMPLE GENERATOR

The example generator is used to generate examples of input files.

The generators fills the sky and load columns with a sequence of numbers as follow

sample	***sky***	***load***
0	0	1
1	2	3
...		
n	2n	2n+1

the FLAG column is filled of zero, the time columns increments of 1 for each couple.

4.5 THE IDL INTERFACE

An IDL interface is provided to allow simple handling of the routine within

4.6 STRUCTURE OF BINARY AND FITS FILES

4.6.1 FITS FILES

FITS files are structured in the same manner of the TOIs files generated by the TMH/TQL..

Data are stored as FITS binary tables with the following structure:

1. minimal primary header
2. extension header
3. binary table

In output the extension header contains the following keywords

Keyword	*Type*	*Meaning*
CREATOR	String	The code creating the file
PFX	String	The prefix for output names
CONTENT	String	QUANTIZED for quantized data, RECONSTRUCTED for reconstructed data, ERROR for processing errors
NAVER	Integer	Naver used in processing
OFFSET	Float	offset_adjust used in processing
QUANT	Float	second_quant used in processing
gmf1	Float	gmf1 used in processing
gmf2	Float	gmf2 used in processing

The binary table is composed of the following columns:

Column	*Name*	*Type*	*Content*
1	OB_TIME	double precision	On board time
2	TIME	double precision	Absolute time
3	sky	either float, double precision or integer	sky
4	load	either float, double precision or integer	load
5	FLAG	integer	The flag

about sky and load, the format depends on the file content. Input files uses double precision, output files containing quantized data (q1, q2) where columns 3 and 4 are integers (1J) the other single precision data.

In output OB_TIME, TIME and FLAG are simply copied from input files.

4.6.2 BINARY FILES

The binary file contains q1, q2 data ordered as following

sample	0	1	2	3	...	2n	2n+1
value	q1(0)	q2(0)	q1(1)	q2(1)	...	q1(n)	q2(n)

values are stored as 16 bits integers, little-endian or big-endian according to the “b” or “l” value of <<output-type>> parameter.

4.7 EXAMPLE OF USE

An example of compilation and use is the script

```
example.sh
```

to be executed by

```
csh example.sh
```

The example runs the quantum_gen_example.x program to generate an example of input and then operates with different combinations of REBA parameters the simulator on the example file.

4.8 WARNING IN ASSIGNING NAVER

In assigning values to Naver consider the following

1. in case the input file comes from the LFI in the form of DTOIs, `Naver` shall have the same value used for the PType 1 acquisition from the instrument;
2. in case the input file comes from the LFI in the form of TOIs, `Naver = 1`;
3. in case the input file comes from the LS which already simulates the coadding of data giving in output averaged data `Naver = 1`.