POLAR
High Level Data Products Format Design Specification

Document No	:	POLAR_TN_410
Issue	:	0.1
Date	:	May 28, 2016

	Name	**Signature**	**Date**
Prepared by	Zhengheng Li		May 20, 2016
Reviewed by			
Aproved by			

Contents

1 Introduction

This chapter contains an introduction to the document "POLAR High Level Data Products Format Design Specification"

1.1 Purpose of the document

Three core pre-processing programs of POLAR SCI and HK raw data have been finished. They are `SCI_Decode`, `HK_Decode` and `Time_Calculate`. For raw data products from POAC, please see the document[1]. `SCI_Decode` is to directly decode 0B level POLAR SCI raw data from POAC, and do time sync at the same time. `HK_Decode` is to directly decode 0B level POLAR HK raw data from POAC, and do some physical value converting work. `Time_Calculate` is to calculate the absolute GPS time of each event in SCI decoded data using the GPS and timestamp sync information in HK decoded data. These three programs are tested by lots of ground data and work well. One important thing is the format or data structure of the output data files. Everyone who uses these programs should know the format and the way of data organization. This document is mainly to clarify the data structure of decoded data produced by the three pre-processing programs.

1.2 Levels of data products

POLAR data products has several different levels. 1M level data is the directly decoded data produced by `SCI_Decode` or `HK_Decode`. It should keep all information in 0B level raw data, and add some auxiliary data which is helpful for data monitor and data analysis later. The level of SCI data after absolute GPS time of each event is calculated and added by `Time_Calculate` is 1P. 1M and 1P level SCI data have almost the same data structure except for absolute GPS time added. HK data does not have 1P level, because 1M level HK data already have absolute GPS time.

One raw data file from POAC could be very big, because it may contain a day of data. The time span of one orbit is about 90 minutes, so it could be convenient to split the data by orbit. The data structure of orbit splitted data should be the same as the data that is not splitted. So, data monitor and data analysis software can directly process the data after and before splitted without any change. The level of orbit splitted data is 1R.

This document will give a clear clarification of data structure of 1M and 1P level SCI decoded data, 1M level HK decoded data. SCI data of one event include one trigger packet and one or more module packets. It is important to understand the data organization of event data in the output ROOT file.

2 Usage of the three programs

Before introducing the data products format, this chapter gives a brief introduction to how to use the three core pre-processing programs.

2.1 Usage of SCI_Decode and HK_Decode

The way of using the two decoding programs `SCI_Decode` and `HK_Decode` are the same, we can run one of them without any command line parameters to see the help information.

Help information of `SCI_Decode` is as following:

```
> SCI_Decode
Usage:
  SCI_Decode [-l <listfile.txt>] [<POL_SCI_data_001.dat> <POL_SCI_data_002.dat> ...]
             [-o <POL_SCI_decoded_data.root>] [-g <POL_SCI_decoding_error.log>]

Options:
  -l <listfile.txt>             text file that contains raw data file list
  -o <decoded_data.root>        root file that stores decoded data
  -g <decoding_error.log>       text file that records decoding error log info

  --version                     print version and author information
```

And help information of `HK_Decode` is as following:

```
> HK_Decode
Usage:
  HK_Decode [-l <listfile.txt>] [<POL_HK_data_001.dat> <POL_HK_data_002.dat> ...]
            [-o <POL_HK_decoded_data.root>] [-g <POL_HK_decoding_error.log>]

Options:
  -l <listfile.txt>             text file that contains raw data file list
  -o <decoded_data.root>        root file that stores decoded data
  -g <decoding_error.log>       text file that records decoding error log info

  --version                     print version and author information
```

There are two ways to input raw data files.

The first way is directly to use command line parameters without options to give file names as following:

```
> SCI_Decode POL_SCI_data_20160517_154345_001.dat POL_SCI_data_20160517_154345_002.dat ...
```

`SCI_Decode` will scan the designated raw data files one by one from left to right and generate only one decoded ROOT file. The default name of the output file is `POL_SCI_decoded_data.root` for `SCI_Decode` if it is not specified by option `-o`.

The second way is to use a text file which contains all the file names line by line. And use option `-l` to input the raw data files. Just as following:

```
> cat listfile.txt
path/to/POL_SCI_data_20160517_154345_001.dat
path/to/POL_SCI_data_20160517_154345_002.dat
path/to/POL_SCI_data_20160517_154345_003.dat
...
> SCI_Decode -l listfile.txt
```

Options `-o` and `-g` are optional. We can use option `-o` to specify the name of output decoded file. If option `-g` is used, `SCI_Decode` and `HK_Decode` will record some log information into a text file, including the raw data of bad packets.

After a run of `SCI_Decode` or `HK_Decode` finished, some counter information will be printed out, including count of total frames and packets, count of CRC error, count and percentage of packets lost, percentage of time aligned event packets, etc.. Such counter information can give some indications of quality of the raw data.

Screen output of `SCI_Decode` is as following:

```
POL_SCI_data_20160517_154345_001.dat
POL_SCI_data_20160517_154345_002.dat
POL_SCI_data_20160517_154345_003.dat
====================================================================================================
total frame count:         783485             total packet count:         17786003
frame invalid count:       0                  - trigger packet count:     8090369
frame invalid percent:     0.00%              - event packet count:       9695515
frame crc error count:     0                  packet invalid count:       65
frame crc error percent:   0.00%              packet invalid percent:     0.00%
frame interruption count:  0                  packet crc error count:     633
frame start error count:   0                  packet crc error percent:   0.00%
total timestamp 0 count:   0                  packet too short count:     291
----------------------------------------------------------------------------------------------------
  ct  mod  >   ped_trig  ped_event   ped_lost  percent    |   noped_trig  noped_event   noped_lost  percent
   1  405  >        766        766          0    0.00%    |       261973       261973            0    0.00%
   2  639  >        766        766          0    0.00%    |       340300       340300            0    0.00%
   3  415  >        765        765          0    0.00%    |       359015       359014            1    0.00%
   4  522  >        758        758          0    0.00%    |       361436       361436            0    0.00%
   5  424  >        763        763          0    0.00%    |       322721       322721            0    0.00%
   6  640  >        763        763          0    0.00%    |       317664       317663            1    0.00%
   7  408  >        760        760          0    0.00%    |       406439       406439            0    0.00%
   8  638  >        757        757          0    0.00%    |       448543       448543            0    0.00%
   9  441  >        758        758          0    0.00%    |       471523       471523            0    0.00%
  10  631  >        758        758          0    0.00%    |       418859       418859            0    0.00%
  11  411  >        769        769          0    0.00%    |       305021       305021            0    0.00%
  12  505  >        757        756          1    0.13%    |       426402       426403           -1   -0.00%
  13  503  >        759        759          0    0.00%    |       495925       495925            0    0.00%
  14  509  >        742        742          0    0.00%    |       519941       519941            0    0.00%
  15  410  >        762        762          0    0.00%    |       420677       420677            0    0.00%
  16  507  >        769        769          0    0.00%    |       321857       321857            0    0.00%
  17  402  >        758        758          0    0.00%    |       392200       392200            0    0.00%
  18  602  >        754        754          0    0.00%    |       506862       506861            1    0.00%
  19  414  >        765        765          0    0.00%    |       482388       482388            0    0.00%
  20  524  >        747        746          1    0.13%    |       437999       437999            0    0.00%
  21  423  >        766        766          0    0.00%    |       246196       246194            2    0.00%
  22  601  >        761        761          0    0.00%    |       365308       365308            0    0.00%
  23  406  >        770        767          3    0.39%    |       326266       325397          869    0.27%
  24  520  >        771        771          0    0.00%    |       402897       402897            0    0.00%
  25  413  >        768        768          0    0.00%    |       317960       317960            0    0.00%
----------------------------------------------------------------------------------------------------
trigg expected sum: 9695404       noped_trigger:   8089584            ped_trigger:     785
event received sum: 9694526       noped_event_sum: 9675499            sec_ped_trigger: 636
total lost percent: 0.01%         mean event rate: 12719 cnts/sec     np_evts per sec: 15213 pkts/sec
transmission rate:  19.96 Mbps    aligned sum:     9675497            aligned percent: 100.00%
====================================================================================================
```

Screen output of `HK_Decode` is as following:

```
POL_HK_data_20160517_154345_001.dat
==========================================================================================
total frame count:          12564              total obox packet count:    6282
frame valid count:          12564              obox valid count:           6281
frame invalid count:        0                  obox invalid count:         1
frame crc passed:           12564              obox crc passed:            6281
frame crc error count:      0                  obox crc error count:       0
frame interruption count:   0
==========================================================================================
```

2.2 Usage of `Time_Calculate`

`Time_Calculate` is used to calculate and add the absolute GPS time of each event in decoded SCI data. It can work only when the GPS time in HK data is valid. We can also run this program without any command line parameters to see the help information.

Help information of `Time_Calculate` is as following:

```
Usage:
  Time_Calculate <POL_SCI_decoded_data.root> -k <POL_HK_decoded_data.root>
                 [-o <POL_SCI_decoded_data_time.root>] [-g <POL_SCI_time_error.log>]

Options:
  -k <hk_decoded_data.root>        root file that stores hk decoded data
  -o <sci_decoded_data.root>       root file that stores sci decoded data after absolute time is added
  -g <time_error.log>              text file that records time calculating error log info

  --version                        print version and author information
```

It is very straightforward. Just use option `-k` to designate the file name of decoded HK data. Options `-o` and `-g` are also optional. Option `-o` is used to specify the file name of the output ROOT file that stores the SCI data after absolute GPS time is added. If option `-o` is not used, the default file name is `POL_SCI_decoded_data_time.root`. When option `-g` is used, this program will record some error log information into a text file.

Screen output of `Time_Calculate` is as following:

```
Copying physical modules data ...
[ ################################################## DONE ]
Calculating time and copying physical trigger data ...
[ ################################################## DONE ]
Copying pedestal modules data ...
[ ################################################## DONE ]
Calculating time and copying pedestal trigger data ...
[ ################################################## DONE ]
================================================================================
phy_error_count: 0 / 8089584         ped_error_count: 0 / 785
================================================================================
```

Absolute GPS time is only added into trigger packets, and all of other data is just copied.

3 Data Structure of ROOT files

This chapter gives a detail explanation of the TTree structure of 1M/1P level SCI data and 1M level HK data. The way of data organization of SCI event data is also clarified in this chapter. It is helpful to know the structure of raw data of SCI and HK first. See chapter 3 (page 13–17) of document[2] to know the frame structure of SCI and HK raw data, and packet structure of HK data. See section 3.4.2 (page 59–64) of document[3] to know the structure of raw HK packet from OBOX. See section 3.4.3 and 3.4.4 (page 65–68, 78–83) of document[3] to know the structure of raw science data packet and trigger data packet. In the ROOT files of decoded data, some data are directly decoded data, and others are auxiliary data that are added or calculated when decoding.

3.1 1M/1P level SCI data

SCI data of 1M level is generated by `SCI_Decode`. There are 4 TTree objects, which store decoded data, and some TNamed objects, which store meta information. Descriptions of them are shown in Table 1

Table 1: Contents of ROOT file of 1M/1P SCI data

Type	**Name**	**Descriptions**
TTree	t_modules	physical modules packets
TTree	t_trigger	physical trigger packets
TTree	t_ped_modules	pedestal modules packets
TTree	t_ped_trigger	pedestal trigger packets
TNamed	m_dattype	string of description of the data type
TNamed	m_version	version of the program that generate this file
TNamed	m_gentime	string of time when this file is generated
TNamed	m_rawfile	list of file names of the raw data
TNamed	m_dcdinfo	some information calculated when decoding

The two TTree objects t_modules and t_trigger are used to store physical event data. t_modules is for module packets from 25 FEEs, and t_trigger is for trigger packets from CT. These two TTree objects are associated by a specific way to match trigger packet and its corresponding module packets of the same physical event. The way of data organization of physical event data will be introduced later. The other two TTree objects t_ped_modules and t_ped_trigger are used to store pedestal event data. Actually, they have exactly the same structure of t_modules and t_trigger. The reason of storing physical events and pedestal events separately is that it is hard to make the order between physical packets and pedestal packets sequencially as time because of the different methods of doing time sync for physical events and pedestal events. After these two kinds of events are stored separately, it is easy to make the order of both trigger and module packets right as time. And it is not hard to iterate all packets (including pedestal and physical, excluding bad) of one module as the order of time by using a global index number. The method will be introduced later.

3.1.1 Contents table of TTree

Here will introduce the data structure of TTree t_modules and TTree t_trigger.

Firstly, contents of TTree t_modules and t_ped_modules are shown in Table 2.

Table 2: Contents of TTree t_modules and t_ped_modules

Type	**Name**	**Descriptions**
Long64_t	trigg_num	Sequential number of the trigger packet of an event.
Long64_t	event_num	Sequential number of the event packet[1] of a module.
Long64_t	event_num_g	Order number of the sequence of appearing in the raw data file.
Int_t	is_bad	if the packet is invalid or has CRC error.
Int_t	pre_is_bad	if the previous packet is invalid or has CRC error.
Int_t	compress	compress mode
Int_t	ct_num	CT number
UInt_t	time_stamp	raw data of TIMESTAMP field of the packet
UInt_t	time_period	overflow counter of time_stamp
UInt_t	time_align	23 LSB of time_stamp
Double_t	time_second	time in seconds from start
Double_t	time_wait	time_second difference since previous event

Next

[1]When I say event packet, it is equal to module packet

Table 2 (Continue)

Type	**Name**	**Descriptions**
Int_t	raw_rate	raw data of RATE field of the packet
UInt_t	raw_dead	raw data of DEADTIME field of the packet
Float_t	dead_ratio	delta(raw_dead) / delta(time_stamp)
UShort_t	status	raw data of the 16 bits STATUS field of the packet
Event_Status_T	status_bit	each bit in status
Bool_t	trigger_bit[64]	raw data of the TRIGGERBIT
Float_t	energy_adc[64]	ADC of energy of the 64 channels
Float_t	common_noise	COMMON NOISE for compress mode 3
Int_t	multiplicity	sum of trigger_bit[64] of this packet

Type Event_Status_T is a C struct. It is used to extract and store each bit of status. Definition of it is shown in Table 3.

Table 3: Definition of struct Event_Status_T

Type	**Name**	**Bit**	**Descriptions**
Bool_t	trigger_fe_busy	15	Flag indicating Frontend Unit is busys.
Bool_t	fifo_full	14	Flag indicating FIFO memory for events is full.
Bool_t	fifo_empty	13	Flag indicating FIFO memory for events is empty.
Bool_t	trigger_enable	12	Flag indicating trigger is enabled.
Bool_t	trigger_waiting	11	Flag indicating FE is waiting for trigger acceptance.
Bool_t	trigger_hold_b	10	Flag indicating HOLD B signal on FE is asserted.
Bool_t	timestamp_enable	9	Flag indicating timestamp is enabled.
Bool_t	reduction_mode_b1	8	bit 1 of Field indicating the reduction mode of the Frontend Unit.
Bool_t	reduction_mode_b0	7	bit 0 of Field indicating the reduction mode of the Frontend Unit.

Next

Table 3 (Continue)

Type	Name	Bit	Descriptions
Bool_t	subsystem_busy	6	Flag indicating one of three subsystems is busy.
Bool_t	dynode_2	5	Flag indicating DYNODE 2 triggered.
Bool_t	dynode_1	4	Flag indicating DYNODE 1 triggered.
Bool_t	dy12_too_high	3	Flag indicating DY12 TOO HIGH triggered.
Bool_t	t_out_too_many	2	Flag indicating T OUT TOO MANY triggered.
Bool_t	t_out_2	1	Flag indicating T OUT 2 triggered.
Bool_t	t_out_1	0	Flag indicating T OUT 1 triggered.

Then, contents of TTree t_trigger and t_ped_trigger are shown in Table 4.

Table 4: Contents of TTree t_trigger and t_ped_trigger

Type	Name	Descriptions
Long64_t	trigg_num	Sequential number of the trigger packet of an event.
Long64_t	trigg_num_g	Order number of the sequence of appearing in the raw data file.
Int_t	is_bad	if the packet is invalid or has CRC error.
Int_t	pre_is_bad	if the previous packet is invalid or has CRC error.
Int_t	type	code of the 4 types of trigger packet
Int_t	packet_num	raw data of packet number of the trigger packet
UInt_t	time_stamp	raw data of Timestamp register of the trigger packet
UInt_t	time_period	overflow counter of time_stamp
UInt_t	time_align	23 MSB of time_stamp
Double_t	time_second	time in seconds from start
Double_t	time_wait	time_second difference since previous event

Next

Table 4 (Continue)

Type	**Name**	**Descriptions**
ULong64_t	frm_ship_time	raw data of the ship time from frame in which this packet is.
ULong64_t	frm_gps_time	raw data of the GPS time from frame in which this packet is.
Long64_t	pkt_start	first entry index of all the adjacent event packets of this event in the modules tree.
Int_t	pkt_count	number of entries of event packets for this event in the modules tree
Int_t	lost_count	number of lost event packets for this event
Int_t	trigger_n	sum of the trigger_bit[64] of all the event packets for this event
UShort_t	status	raw data of Status register of the trigger packet
Trigg_Status_T	status_bit	each bit in status
UChar_t	trig_sig_con[25]	raw data of Trigger signals conditions for each frontend
Trig_Sig_Con_T	trig_sig_con_bit	each bit in trig_sig_con[25] for each frontend
Bool_t	trig_accepted[25]	raw data of FEE TRIGGER ACCEPTED for each frontend
Bool_t	trig_rejected[25]	raw data of FEE TRIGGER REJECTED for each frontend
UInt_t	raw_dead	raw data of the dead time counter field
Float_t	dead_ratio	delta(raw_dead) / delta(time_stamp) / 4
Int_t	abs_gps_week*	week of absolute gps time of this event.
Double_t	abs_gps_second*	second of absolute gps time of this event.
Bool_t	abs_gps_valid*	if the absolute gps time is valid.

Type Trigg_Status_T and Trig_Sig_Con_T are C structs. They are used to extract and store each bit of status and trig_sig_con[25] respectively. Definitions of the two struct types are shown in Table 5 and Table 6 respectively.

Table 5: Definition of struct Trigg_Status_T

Type	Name	Bit	Descriptions
Bool_t	science_disable	15	Flag indicating the science packets generation by Central Trigger Unit is disabled.
Bool_t	master_clock_enable	14	Flag indicating the Master Clock generation is enabled.
Bool_t	saving_data	13	Flag indicating the science packet is being stored in FIFO.
Bool_t	taking_event_or_ped	12	Flag indicating the Central Trigger Unit state machine is doing the event or pedestal acquisition.
Bool_t	fifo_full	11	Flag indicating FIFO in Central Processing Unit is full.
Bool_t	fifo_almost_full	10	Flag indicating FIFO in Central Processing Unit is almost full.
Bool_t	fifo_empty	9	Flag indicating FIFO in Central Processing Unit is empty.
Bool_t	fifo_almost_empty	8	Flag indicating FIFO in Central Processing Unit is almost empty.
Bool_t	any_waiting	7	Flag indicating at least one FEE sent the WAITING signal to Central Processing Unit.
Bool_t	any_waiting_two_hits	6	Flag indicating at least one FEE, that has two hits, sent the WAITING signal to Central Processing Unit.
Bool_t	any_tmany_thigh	5	Flag indicating at least one FEE, that has Too Many or Too High flags set, sent the WAITING signal to Central Processing Unit.
Bool_t	packet_type_b2	4	bit 2 of Field indicating the type of science packet being processed by the state machine of Central Trigger Unit.

Next

Table 5 (Continue)

Type	Name	Bit	Descriptions
Bool_t	packet_type_b1	3	bit 1 of Field indicating the type of science packet being processed by the state machine of Central Trigger Unit.
Bool_t	packet_type_b0	2	bit 0 of Field indicating the type of science packet being processed by the state machine of Central Trigger Unit.

Table 6: Definition of struct Trig_Sig_Con_T

Type	Name	Bit	Descriptions
Bool_t	fe_busy[25]	5	Flag indicating the status of the FE BUSY signal from this Frontend Unit.
Bool_t	fe_waiting[25]	4	Flag indicating the status of the FE WAITING signal from this Frontend Unit.
Bool_t	fe_hold_b[25]	3	Flag indicating the status of the FE HOLD B signal from this Frontend Unit.
Bool_t	fe_tmany_thigh[25]	2	Flag indicating the status of the FE TMANY THIGH signal from this Frontend Unit.
Bool_t	fe_tout_2[25]	1	Flag indicating the status of the FE TOUT 2 signal from this Frontend Unit.
Bool_t	fe_tout_1[25]	0	Flag indicating the status of the FE TOUT 1 signal from this Frontend Unit.

3.1.2 Directly decoded data in t_modules

Some data in t_modules is directly decoded from module packet without any change. Here list and explain all of them.

compress Bit [8:7] of module status word. It is the code of reduction mode. There are four different reduction mode types. 0 is for default mode, 1 is for simple mode, 2 is for pedestal mode, and 3 is for full reduction mode.

ct_num This is the CT number, raw data of FEE Unit number. The range of it is from 1 to 25, indicating which module this packet is from.

time_stamp Raw data of TIMESTAMP field of this packet. The number of valid bits is 24. The unit of it is $40.96\mu s$.

raw_rate Raw data of RATE word of this packet.

raw_dead Raw data of DEADTIME word of this packet. The unit of it is the same as TIMESTAMP.

status Raw data of module STATUS word.

status_bit This is a C struct of pure bool type. Each bit of STATUS word is extracted and stored in this struct respectively. Names of the fields indicate the meaning of each bit.

trigger_bit[64] Array of each bit of TRIGGERBIT. The type of it is Bool_t. True means the corresponding channel is triggered.

energy_adc[64] Array of ADC of each channel. For mode 2 and mode 3, some channels have no ADC data, in this case, ADC of the channel is 0. ADC of mode 3 is special. The output ADC of mode 3 is $(ADC_{raw}-2048)\times 2$. After this calculation, the unit of ADC of mode 3 is the same as other modes. One important thing is that ADC of mode 3 is already pedestal subtracted in firmware, but common noise is not subtracted. The common noise for mode 3 is stored in common_noise. For mode 0, 1 and 2, the output ADC is equal to the raw ADC.

common_noise Raw data of COMMON NOISE for mode 3 is subtracted by 2048. The reason to subtract 2048 is that firmware add an extra 2048 to common noise. For ohter compress modes, the value of common_noise is 0.

3.1.3 Auxiliary data in t_modules

One important fact is that packets of a specific module and trigger packets are ordered exactly as time in the raw data file. It is better to add some auxiliary data related to the sequence of time that is helpful for data monitor and data analysis later. All of the auxiliary data in t_modules is listed and explained here.

trigg_num This is the sequential number of trigger packet of this event. Module packets which belong to the same event have the same trigg_num. This number is used for organization of event data. It start from 0. It is -1 if this module packet has no corresponding trigger packet and -2 when this module packet is bad.

event_num Sequential number of module packets. This number for different modules is independent. This number is added when saving data into TTree. In ohter words, this number is also independent for pedestal and physical packets. This number is continuous and incremental for a specific module in the same TTree, t_modules or t_ped_modules. It start from 0, and it is -1 when this packet is bad.

event_num_g Order number of the sequence of appearing in the raw data file. This number for different modules is independent. The difference between this number and event_num is that this number is added when scanning the raw data file. It counts both pedestal and physical packets. Because pedestal and physical packets are stored in different TTree, this number is incremental but sometimes discontinuous for a specific module in the same TTree. It start from 0, and it is -1 when this packet is bad. This number is used for iterating all packets of one module including pedestal and physical as the order of time.

is_bad An integer value that indicates whether this packet is bad. The value is 3 when this packet is too short, 2 when invalid, 1 when CRC error, and 0 when good. The value is -1 when this packet is good but the timestamp is 0.

pre_is_bad Value of is_bad of the previous packet of the same module. This value is necessary because if the previous packet is bad, some ohter auxiliary data in t_modules such as time_wait, dead_ratio is unknown and wrong.

time_period time_stamp of module packet will overflow about every 11.45 minutes. This value records the total number of overflow from start.

time_align It is the 23 LSB of time_stamp of this packet. This is useful for time alignment. It is the counterpart of 23 MSB of time_stamp of trigger packet, that is the time_align of trigger packet. time_align of both module packet and trigger packet have the same time unit and range.

time_second This is the time in second unit from start. It is equal to $(\text{time\_period} \times 2^{24} + \text{time\_stamp}) \times 40.96 \times 10^{-6}$

time_wait This is the difference of time_second between this packet and previous packet of the same module. The unit of it is second.

dead_ratio Ratio of the increment of raw_dead to the increment of time_stamp. The increment is between this packet and previous packet of the same module. In formula, $\text{dead\_ratio} = \Delta(\text{raw\_dead})/\Delta(\text{time\_stamp})$.

multiplicity Sum of array trigger_bit[64] of this packet. It indicates how many bars of this module is fired.

3.1.4 Directly decoded data in t_trigger

Here list and explain the directly decoded data of trigger packet in TTree t_trigger.

type Code of the 4 types of trigger packet. 0x00F0 is for pedestal event, 0x00FF is for normal event, 0xF000 is for prescale single event, and 0xFF00 is for cosmic event.

packet_num Raw data of the packet number word of this trigger packet.

time_stamp Raw data of the timestamp register double word of this trigger packet. The number of valid bits is 32. The unit of it is $80ns$.

frm_ship_time Raw data of the 6 bytes ship time decoded from the header of frame in which this trigger packet is.

frm_gps_time Raw data of the 6 bytes GPS time decoded from the header of frame in which this trigger packet is.

status Raw data of the status register word of this trigger packet.

status_bit This is a C struct of pure bool type which is used to extract and store each bit of status respectively. Names of the fields indicate the meaning of each bit.

trig_sig_con[25] Array to store the byte of trigger signals conditions for each frontend.

trig_sig_con_bit C struct to extract and store each bit of trigger signals conditions for each frontend respectively. Each field of this C struct is of type bool[25].

trig_accepted[25] Array to store each bit of FEE TRIGGER ACCEPTED for each frontend respectively.

trig_rejected[25] Array to store each bit of FEE TRIGGER REJECTED for each frontend respectively.

raw_dead Raw data of the dead time counter word of this trigger packet.

3.1.5 Auxiliary data in t_trigger

Some of the auxiliary data added in t_trigger is similar to that is added in t_modules. Some are used to organize the event data, that is, to find the corresponding module packets of the same event in t_modules. The three GPS related branches with a star tagged shown in Table 4 are added by program `Time_Calculate`. They belong to the 1P level SCI data, and the 1M level SCI data does not have these three branches. All the auxiliary data in t_trigger is listed and explained below.

trigg_num Sequential number of trigger packet. Like event_num of t_modules, this number is independent for pedestal and physical trigger packets and is always continuous and incremental in the same TTree, t_trigger or t_ped_trigger. It start from 0, and it is -1 when this trigger packet is bad.

trigg_num_g Order number of the sequence of appearing in the raw data file. Also similar to event_num_g for module packets, this number is added when scanning the raw data file and counts both pedestal and physical trigger packets. Because pedestal and physical trigger packets are stored in different TTree, This number is incremental but sometimes discontinuous in the same TTree. It start from 0, and it is -1 when this trigger packet is bad.

is_bad The same as that in t_modules but for trigger packets.

pre_is_bad The same as that in t_modules but for trigger packets.

time_period The same as that in t_modules but for trigger packets. time_stamp of trigger packet will overflow about every 5.73 minutes.

time_align It is the 23 MSB of time_stamp of this trigger packet. It is also useful for time alignment. It is the counterpart of 23 LSB of time_stamp of module packet.

time_second Time in second unit from start when this trigger packet is generated. It is equal to $(\text{time\_period} \times 2^{32} + \text{time\_stamp}) \times 80 \times 10^{-9}$.

time_wait The same as that in t_modules but for trigger packets.

pkt_start This is the branch that records the first entry index of the module packets in t_modules for the same event. In t_modules, module packets that belong to the same event are adjacent to each other. The total number of module packets for the same event is recorded by pkt_count. The value of pkt_start is -1 when this trigger packet loses all event packets, and -2 when this trigger packet is bad.

pkt_count This is the branch that records the total number of module packets in t_modules that belong to the same event as this trigger packet. We can use pkt_start and pkt_count to find all the corresponding module packets of this trigger packet in t_modules.

lost_count An integer value that indicates how many packets this event loses. There are two reasons for the lost. The first is that it is realy lost. And the second is that some module packets failed to time align with this trigger packet because of possible timestamp issue.

trigger_n Sum of the trigger_bit[64] of all the module packets of this event. It is the number of how many bars are fired in this event. It may includes several modules.

dead_ratio It should be the same as that of t_modules, but there is a difference. For trigger packets, $\text{dead\_ratio} = pre\Delta(\text{raw\_dead})/\Delta(\text{time\_stamp})$. That means $\text{dead\_ratio}_3 = \frac{\text{raw\_dead}_2 - \text{raw\_dead}_1}{\text{time\_stamp}_3 - \text{time\_stamp}_2}$.

abs_gps_week Added by `Time_Calculate`. It is the week of absolute GPS time when this event occurred.

abs_gps_second Added by `Time_Calculate`. It is the second of absolute GPS time when this event occurred.

abs_gps_valid Added by `Time_Calculate`. It indicates if the absolute GPS time is valid.

3.1.6 Iterating pedestal and physical packets together

Because of the different mehtods of doing time alignment for pedestal and physical packets, if save both pedestal packets and physical packets in the same TTree, it is hard to make the sequence between pedestal and physical packets as the order of time. But it is easy to make the sequence of only pedestal or physical packets as the order of time in the same TTree. So I have to save pedestal and physical data separately. But sometimes it is needed to iterate pedestal and physical packets together as the order of time. We can use event_num_g in both t_modules and t_ped_modules, and trigg_num_g in both t_trigger and t_ped_trigger to do this thing easily. event_num_g (for module packets) and trigg_num_g (for trigger packets) are added when scanning the raw data file. They are the order of appearing in raw data file of the packet. Pedestal and physical packets for the same module use the same counter. In pedestal and physical TTree, event_num_g (or trigger_num_g) looks like Table 7.

Table 7: number in t_modules and t_ped_modules

t_ped_modules		**t_modules**	
event_num	event_num_g	event_num	evnet_num_g
0	0		
1	1		
2	2		
		0	3
		1	4
		2	5
		3	6
3	7		
		4	8
		5	9
4	10		
		6	11
		7	12
		8	13

Notice that event_num and event_num_g are all independent for different modules. Table 7 just shows the case of only a specific module.

It is clear that we can open the two TTree t_ped_modules and t_modules at the same time and iterate both pedestal and physical packets of one specific module together as the order of time by comparing event_num_g. For trigger packets, the method is the same, by comparing trigg_num_g.

Notice that event_num is “local” in the same TTree t_ped_modules or t_modules, but event_num_g is “global” between the two TTree t_ped_modules and t_modules. That is what the suffix “_g” mean.

3.1.7 Organization of event data

Data of an event (both pedestal and physical) contains one trigger packet and one or more module packets. Packets of trigger and module have different data structure, so they must be stored in different TTree. But the matching between trigger packet and module packet of the same event is also important. For one trigger packet in t_trigger, we have to know which module packets in t_modules are the corresponding module packets of the same event. And on the contrary, for one module packet in t_modules, we have to know which trigger packet in t_trigger is the corresponding trigger packet of the same event. Six extra branches in t_trigger and t_modules are used to do this kind of data organization. They are trigg_num, pkt_start, pkt_count, lost_count in trigger packet, and trigg_num, event_num in t_modules. The relationship among them is shown in Table 8.

Table 8: organization of event data between t_trigger and t_modules

t_trigger				**t_modules**			
entry	trigg_num	pkt_start	pkt_count	entry	trigg_num	ct_num	event_num
0	0	0	2	0	0	2	0
				1	0	3	0
1	1	2	3	2	1	8	0
				3	1	7	0
				4	1	2	1
2	2	5	1	5	2	3	1
3	3	6	2	6	3	6	0
				7	3	7	1
4	4	8	2	8	4	3	2
				9	4	4	0
5	5	10	3	10	5	12	0
				11	5	7	2
				12	5	8	1
6	6	13	1	13	6	3	3
7	7	14	2	14	7	2	2
				15	7	3	4

The organization of event data between t_trigger and t_modules is shown clearly in Table 8. There are two key rules: 1) trigger packet in TTree t_trigger records the position of modules packets in TTree t_modules using pkt_start and pkt_count; 2) module packet in TTree t_modules records the trigg_num of the trigger packet in TTree t_trigger of the same event. trigg_num is unique for each trigger packet in t_trigger. event_num of module packets in t_modules is independent for different modules, but unique for a specific module.

This organization looks some complicated. But it has some advantages comparing just storing the ADC data of one event into a 1600 array. Because the packet data is stored module by module, all data in module packet and trigger packet can be kept. And it is helpful for data analysis, because some steps of the analysis chain, such as pedestal subtraction, crosstalk correction and energy calibration, are always done module by module. Only the calculation of scattering angle is done in the whole instrument scope. Though, the disadvantage is that we have to keep the structure not changed when we do pedestal subtraction, crosstalk correction and energy calibration etc. before doing the calculation of scattering angle.

When we want to merge the module data of each event into a 1600 array, or when we calculate the scattering angle directly, we must iterate the data event by event. The best way to do this is as following.

```
// value declaration.
t_trigger->SetBranchAddress("pkt_start", pkt_start);
t_trigger->SetBranchAddress("pkt_count", pkt_count);
t_trigger->SetBranchAddress("lost_count", lost_count);
t_trigger->SetBracnhAddress("is_bad", trigg_is_bad);
// SetBranchAddress for other branches in t_trigger
t_modules->SetBranchAddress("ct_num", ct_num);
// SetBranchAddress for other branches in t_modules
for (Long64_t i = 0; i < t_trigger->GetEntries(); i++) {
    t_trigger->GetEntry(i);
    // when lost_count > 0, this event may lose module packets
    if (trigg_is_bad > 0 || lost_count > 0)
        continue;
    // process trigger packet data of this event.
    for (Long64_t j = pkt_start; j < pkt_start + pkt_count; j++) {
        t_modules->GetEntry(j);
        // process each module packet data of this event,
        // merge them into a 1600 array or calculate scattering angle directly.
        // use ct_num to identify which module this packet is from.
        // do not need to check if this module packet is bad,
        // because bad module packet can not be here.
    }
}
```

In addition, the lost_count branch records how many module packets this event loses. When lost_count is not 0, this event is bad, because it loses information.

3.2 1M level HK data

HK data of 1M level is generated by `HK_Decode`. There are 2 TTree objects, which store decoded data, and some TNamed objects, which store meta information. Descriptions of them are shown in Table 9.

Table 9: Contents of ROOT file of 1M HK data

Type	Name	Descriptions
TTree	t_hk_obox	obox housekeeping packets
TTree	t_hk_ibox	ibox housekeeping info
TNamed	m_dattype	string of description of the data type
TNamed	m_version	version of the program that generate this file
TNamed	m_gentime	string of time when this file is generated
TNamed	m_rawfile	list of file names of the raw data

The two TTree objects t_hk_obox and t_hk_ibox are used to store the decoded HK packets from IBOX and OBOX respectively. HK packet from OBOX is generated with a periodicity of 2 seconds. And it will be divided into two HK packets in IBOX, and packed into odd and even packets. t_hk_obox is to store the whole HK packet from OBOX after the two half packets divided into odd and even packets are combined. t_hk_ibox is to store the other information except for that of OBOX HK packet in an IBOX HK packet, such as command feedback from OBOX.

3.2.1 Contents table of TTree

Like SCI data, here will give the table of contents of the two TTree objects first.

Firstly the contents of TTree t_hk_obox are shown in Table 10.

Table 10: Contents of TTree t_hk_obox

Type	Name	Descriptions
Int_t	odd_index	Frame Index of odd packet, -1 when lost.
Int_t	even_index	Frame Index of even packet, -1 when lost.
Int_t	odd_is_bad	code that indicates if the odd half packet is bad. 3 when lost, 2 when invalid, 1 when crc error, 0 when good.

Next

Table 10 (Continue)

Type	Name	Descriptions
Int_t	even_is_bad	code that indicates if the even half packet is bad. 3 when lost, 2 when invalid, 1 when crc error, 0 when good.
UShort_t	packet_num	raw data of the Packet number word
UInt_t	timestamp	raw data of the OBOX time-stamp double word
UChar_t	obox_mode	raw data of the 4 bits OBOX operational mode
UShort_t	cpu_status	raw data of the 12 bits OBOX CT CPU status
UChar_t	trig_status	raw data of the OBOX CT Trigger status byte
UChar_t	comm_status	raw data of the OBOX CT Communication status byte
Float_t	ct_temp	physical value of the Central Trigger temperature
Float_t	chain_temp	physical value of the Sensor chain temperature
UShort_t	reserved	raw data of the Reserved word
UShort_t	lv_status	raw data of the LV power supply status word
UInt_t	fe_pattern	raw data of the FEs powered double word
Float_t	lv_temp	physical value of the LV power supply temperature
UShort_t	hv_pwm	raw data of the HV PWM setting word
UShort_t	hv_status	raw data of the HV power supply status word
UShort_t	hv_current[2]	raw data of the HV current readout1 and readout2 words
UChar_t	fe_status[25]	raw data of the Module status byte of each module
Float_t	fe_temp[25]	physical value of the Module temperature of each module

Next

Table 10 (Continue)

Type	Name	Descriptions
Float_t	fe_hv[25]	physical value of the HV voltage setting of each module
Float_t	fe_thr[25]	physical value (if packet_num % 4 == 0) or raw data (if packet_num % 4 != 0) of the Threshold setting of each module
UShort_t	fe_rate[25]	raw data of the Count rate word of each module
UShort_t	fe_cosmic[25]	raw data of the Too many/too high rate word of each module
Float_t	flex_i_p3v3[5]	physical value of the Current at P3V3 rail of each FLEX
Float_t	flex_i_p1v7[5]	physical value of the Current at P1V7 rail of each FLEX
Float_t	flex_i_n2v5[5]	physical value of the Current at N2V5 rail of each FLEX
Float_t	flex_v_p3v3[5]	physical value of the Voltage at P3V3 rail of each FLEX
Float_t	flex_v_p1v7[5]	physical value of the Voltage at P1V7 rail of each FLEX
Float_t	flex_v_n2v5[5]	physical value of the Voltage at N2V5 rail of each FLEX
Float_t	hv_v_hot	physical value of the Voltage at HV Hot P3V3 rail
Float_t	hv_i_hot	physical value of the Current at HV Hot P3V3 rail
Float_t	ct_v_hot[2]	physical value of the Voltage at CT Hot P3V3 and 1V5 rail
Float_t	ct_i_hot[2]	physical value of the Current at CT Hot P3V3 and 1V5 rail
Float_t	hv_v_cold	physical value of the Voltage at HV Cold P3V3 rail
Float_t	hv_i_cold	physical value of the Current at HV Cold P3V3 rail
Float_t	ct_v_cold[2]	physical value of the Voltage at CT Cold P3V3 and 1V5 rail

Next

Table 10 (Continue)

Type	**Name**	**Descriptions**
Float_t	ct_i_cold[2]	physical value of the Current at CT Cold P3V3 and 1V5 rail
UInt_t	timestamp_sync	raw data of the word of Time-stamp at last sync
UShort_t	command_rec	raw data of the byte of Command received counter
UShort_t	command_exec	raw data of the byte of Command executed counter
UShort_t	command_last_num	raw data of the word of Command last executed number
UShort_t	command_last_stamp	raw data of the word of Command last executed time-stamp
UShort_t	command_last_exec	raw data of the word of Command last executed code
UShort_t	command_last_arg[2]	raw data of the words of Command last executed argument1 and argument2
UShort_t	obox_hk_crc	raw data of the word of OBOX HK packet CRC
UShort_t	saa	raw data of the 2 bits SAA flat
UShort_t	sci_head	raw data of the word of OBOX science packet header counter
ULong64_t	gps_pps_count	raw data of the 8 bytes Time_PPS
ULong64_t	gps_sync_gen_count	raw data of the 8 bytes Time_synchComGen
ULong64_t	gps_sync_send_count	raw data of the 8 bytes Time_synchComTx
ULong64_t	ibox_gps	raw data of the 6 bytes IBOX GPS time of odd packet
Int_t	abs_gps_week	the GPS week of ibox_gps of this packet
Double_t	abs_gps_second	the GPS second of ibox_gps of this packet

Secondly, the contents of TTree t_hk_ibox are shown in Table 11.

Table 11: Contents of TTree t_hk_ibox

Type	Name	Descriptions
Int_t	frame_index	Frame Index of this packet
Int_t	pkt_tag	Packet tag of this packet
Int_t	is_bad	code that indicates if this packet is bad. 2 when invalid, 1 when CRC error, and 0 when good.
ULong64_t	ship_time	raw data of the 6 bytes Ship time of this packet
UShort_t	error[2]	raw data of the two words of Command feedback error number1 and number2
UShort_t	frame_head	raw data of the Frame header word of the 6 words command feedback from OBOX
UShort_t	command_head	raw data of the Command frame header of the 6 words command feedback from OBOX
UShort_t	commnad_num	raw data of the Command number word of the 6 words command feedback from OBOX
UShort_t	command_code	raw data of the Command code word of the 6 words command feedback from OBOX
UShort_t	command_arg[2]	raw data of the two words of Command argument1 and argument2 of the 6 words command feedback from OBOX
ULong64_t	ibox_gps	raw data of the 6 bytes IBOX GPS time of this packet
Int_t	abs_gps_week	the GPS week of ibox_gps of this packet

Next

Table 11 (Continue)

Type	Name	Descriptions
Double_t	abs_gps_second	the GPS second of ibox_gps of this packet

3.2.2 Conversion of physical value

The meaning of each branch in the two TTree t_hk_obox and t_hk_ibox is explained clearly in Table 10 and Table 11. There is no need to give more detailed information of them in this document. What we concern more about is how the physcial values are converted from raw data. Notice that all branches in t_hk_obox with type Float_t are physical value converted. Here will give the formula of each of them.

ct_temp raw data of it is one word.
ct_temp = raw_data[11:4] > 0x7F ? raw_data[11:4] - 2 * 0x80 : raw_data[11:4]

chain_temp raw data of it is one word.
chain_temp = raw_data[11:4] > 0x7F ? raw_data[11:4] - 2 * 0x80 : raw_data[11:4]

lv_temp raw data of it is one word.
lv_temp = 27 + ((raw_data - 0x8000) / 16384.0 * 2.5 / 2 - 28E-3) / 93.5E-6

fe_temp[25] raw data of it is one byte for each module.
fe_temp = raw_data > 0x7F ? raw_data - 2 * 0x80 : raw_data

fe_hv[25] raw data of it is one word for each module.
fe_hv = raw_data[15:4] * 0.303

fe_thr[25] raw data of it is one word for each module.
fe_thr = packet_num % 4 == 0 ? raw_data / 4096.0 * 3.5 - 2.0 : raw_data

flex_i_p3v3[5] raw data of it is one word.
flex_i_p3v3 = (raw_data - 0x8000) / 104.858

flex_i_p1v7[5] raw data of it is one word.
flex_i_p1v7 = (raw_data - 0x8000) / 104.858

flex_i_n2v5[5] raw data of it is one word.
flex_i_n2v5 = (raw_data - 0x8000) / 104.858

flex_v_p3v3[5] raw data of it is one word.
flex_v_p3v3 = (raw_data - 0x8000) / 4681.14

flex_v_p1v7[5] raw data of it is one word.
flex_v_p1v7 = (raw_data - 0x8000) / 8426.06

flex_v_n2v5] raw data of it is one word.
flex_v_n2v5 = (raw_data - 0x8000) / (-2407.44) + flex_v_p3v3

hv_v_hot raw data of it is one word.
hv_v_hot = (raw_data - 0x8000) / 4681.14

hv_v_cold raw data of it is one word.
hv_v_cold = (raw_data - 0x8000) / 4681.14

hv_i_hot raw data of it is one word.
hv_i_hot = (raw_data - 0x8000) / 104.858
hv_i_cold raw data of it is one word.
hv_i_cold = (raw_data - 0x8000) / 104.858
ct_v_hot[2] raw data of it is one word.
ct_v_hot[0] = (raw_data - 0x8000) / 4681.14
ct_v_hot[1] = (raw_data - 0x8000) / 9011.20
ct_v_cold[2] raw data of it is one word.
ct_v_cold[0] = (raw_data - 0x8000) / 4681.14
ct_v_cold[1] = (raw_data - 0x8000) / 9011.20
ct_i_hot[2] raw data of it is one word.
ct_i_hot[0] = (raw_data - 0x8000) / 104.858
ct_i_hot[1] = (raw_data - 0x8000) / 104.858
ct_i_cold[2] raw data of it is one word.
ct_i_cold[0] = (raw_data - 0x8000) / 104.858
ct_i_cold[1] = (raw_data - 0x8000) / 104.858

One thing should be noticed. Some data fields in OBOX HK packet provide multiple information, such as the Threshold setting. Only when packet_num % 4 == 0, this field is the threshold value, it should be converted. When packet_num % 4 == 1, 2 or 3, this field is some other counters, and it should not be converted. See the housekeeping packet structure in document[3] for more detail about this.

4 About splitting data by orbit

After POLAR is in orbit, one file from POAC can be very big, because it may contains the data of one day. The size of one data file can be several GBs. It must be not convenient to analyze so big a file at a time. It is necessary to split the big decoded file into some small files. That may be a good way to split file by orbit, because the orientation of POLAR, space environment, etc. are all changing periodically between differnt orbits. The level of orbit splitted data is 1R.

4.1 Some rules to obey when splitting data by orbit

The orbit splitting program is not started yet. But here will list some rules that are planned to obey when splitting data by orbit.

1. The data structure of 1R level data (orbit splitted) should be the same as 1P level for SCI data and 1M level for HK data. So that the data monitor and data analysis software can apply to the data before and after splitted without any change.
2. Some index numbers, such as trigg_num, trigg_num_g, event_num, event_num_g and pkt_start in trigger packet should be rearranged. They all should be started from 0 in the small orbit splitted data file. time_second and time_period should also be started from 0 after splitted.
3. The data is planned to split by orbit according to GPS time when TG-2 pass through the start point of the orbit. We can get the time from the platform information in 1553B data.
4. A global orbit number should be added into each of the orbit splitted data files.

References

[1] POLAR_space_data_from_GESSA/POAC data products.pdf

[2] POLAR_data_link/Introduction_of_POLAR_data_link.pdf

[3] TN_318/POLAR_OBOX_Software_Design_Specification.pdf