ProtoDUNE SSP software test plan

Revision 1.4
20170613 JTA

Introduction

This document describes the plan for testing production quantities of the ProtoDUNE SiPM Signal Processor (SSP), circuit board revision 17PC001A. Testing of these modules is viewed as occurring in three phases:

I. Post-assembly basic checkout, a manual procedure with little software involvement
II. Main testing using a minimal software interface LabWindows program that guides the user through the test procedure and displays results but does not allow manual control;
III. Final checkout testing using a set of Linux programs based upon the DUNE DAQ system.

Tracking of a module's progress through the latter two phases is recorded and archived using electronic records. A 'checklist' document in PDF with data entry fields is used for the first section.

Relationship of test software to the "LBNEWare" engineering software

The test software used for main testing is fully integrated with "LBNEWare" as a single software program. When the software starts a sign-in screen similar to that sketched in Figure 1 is presented.

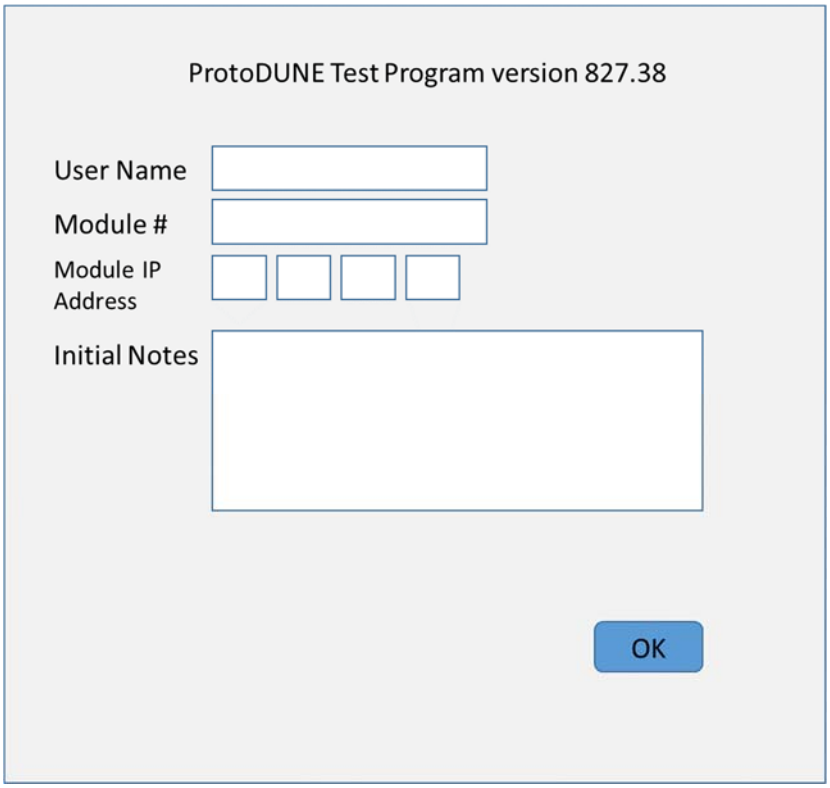

Figure 1 - Initial sign on screen

The user is required to enter a non-blank user name, module serial number and IP address, but the notes field is optional. Upon hitting 'OK' the program will attempt to connect with the module at the IP address given, and if successful, check the module serial number entered against the data stored in the Flash RAM of the module. If the serial number matches the program proceeds into the test sequence. If the serial number does not match and/or communication fails the program will flag the error and refuse to advance.

It will be possible to enter the original LBNEWare software by entering the user name "LBNEWare". This will allow engineers and power users to diagnose failures and/or test new versions of firmware as needed. Entry of any other non-blank user name will cause the program to stay in "test mode", during which access to the LBNEWare screens will be blocked.

Test Mode Main Screen

The main screen for test mode presents a list of tests that can be performed upon a given module. Each test is a stand-alone sub-program that performs all necessary initialization and prompting of the user such that tests may be run in any order.

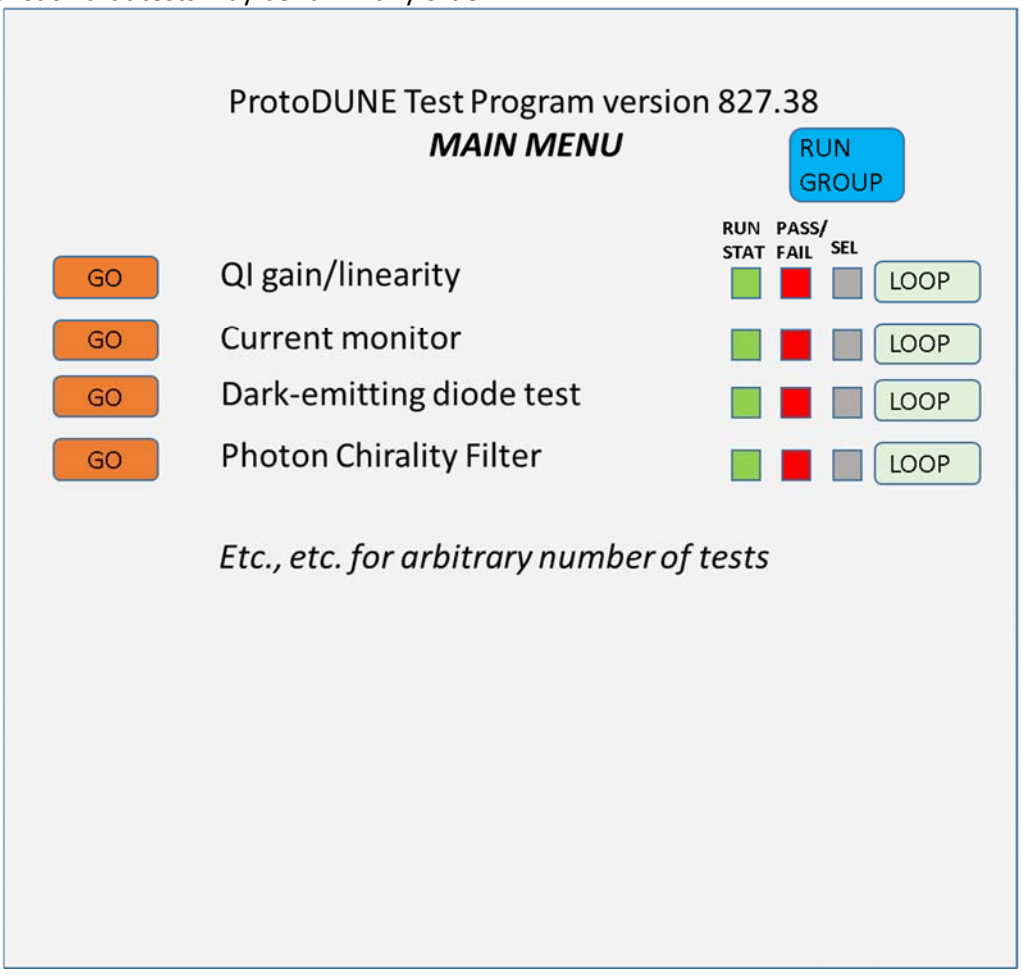

Figure 2 - Testing main menu

Each test is individually activated by hitting the 'GO' button for the test. The 'RUN STAT' indicator provides visual feedback that the test has or has not yet been run for this module. The 'PASS/FAIL' indicator updates each time the test is run, illuminating green if the test passed, red if the test failed.

The ‘PASS/FAIL’ indicators initialize to black and do not take on a color after the test has been run. The ‘SEL’ controls allow the user to select a sequence of tests that will be run automatically in order when the ‘RUN GROUP’ button is hit. Wherever possible, a ‘LOOP’ button will be provided that places the test into *scope loop* mode. In *scope loop* mode the test is run over and over until halted by the user, but no checking of module data or analysis of results is performed. The *scope loop* mode only performs the setup and running of the test to allow the experienced technician or engineer to poke around with an oscilloscope to look for assembly flaws.

Individual test screens

Each test accessible from the main screen will pop up its own screen(s) specific to the test. The purpose of this subsidiary screen is to ensure that all external physical connections necessary to run the test are made. The user shall be prompted for each physical connection required and shall be required to answer ‘ok’ to each prompt before the next prompt appears. When all user interaction is complete the program will close the setup screen and then perform the automated test.

Except when in *scope loop* mode, a separate window shall provide test progress feedback, measurement results and general pass/fail status. This summary information will vary from test to test but may be as simple as a list of tests performed or as complicated as a graphical display with error bands. If the test has been invoked as a *scope loop* the summary screen will be replaced by a generic screen indicating that looping is in progress until the user hits the “DONE” button.

Upon completion of the test the summary screen will linger until the user hits an OK button, unless the test is one of a set initiated by the **Run Group** button on the main screen. In **Run Group** mode the data shall still be logged to file but the summary screen shall not display; rather, upon completion of the test the main program shall immediately advance to the next test in the group that shall display its setup screen like usual. At the end of a **Run Group** sequence, when the last test is completed, a generic screen indicating that the group run is complete shall be displayed until the user hits the “OK” button.

Specific details of scope loop mode

When in the scope loop mode, the test shall run through the external setup sequence once and then enter an infinite loop during which the test is run in identical fashion to normal, but the saving of data to disk is disabled. At the end of each run of the test the module shall be reset and re-initialized such that each iteration of the test provides the same sequence of operations with no history carried in the module from one run to the next. Scope loop mode is intended for probing use only. Collection of statistics from many runs of the same test on the same module is deemed a sub-function of the **Run Group** mode.

Run Group response to test results

The normal **Run Group** operation will be to run the selected tests in order, once per test, until a failure is reported; At that point **Run Group** aborts early. Options will be provided either through a configuration file or top-level control to allow the group to be run with errors ignored, and also to allow the group to be run ‘n’ times instead of once, to allow for statistical collection. Test subprograms shall be provided with input flags alerting them to whether **Run Group** mode is in effect, along with the error response and the current loop number of a loop of ‘n’ groups, so that the verbosity of file output and/or synchronization to the Box cloud drive may be appropriately managed.

Data storage

A folder structure on the Box cloud storage server will be used to save all test results. This will be accomplished by using the Box Sync application on any PC used with the test program. Box Sync runs in the background and monitors a specified folder on the local machine. Whenever there are changes to the monitored folder, the application automatically re-copies the monitored folder to the cloud folder, maintaining revision control information as it does so. To avoid excessive copying and versioning overhead, each individual test sequence shall store the data to a temporary file in a non-monitored folder of the local machine and append the test data to the monitored folder only when the specific test is complete. Thus the version control trace of Box will show one revision per execution of a test, resulting in the ability to extract not only how many times a test has been run but also when those tests were run.

Synchronization of local data with Box

All data written by each individual test shall be written to a local temporary file on the hard drive of the machine running the test program. Normally, at the end of each individual test, the local temporary file shall be appended to the shared folder. The background Box Share application monitors the shared folder and typically initiates a backup to Box (with version information) within one second.

Data stored by each test

Upon execution each test, a single **summary record** is written to an ever-growing **global history file** as defined below. The global history file is formatted in comma-separated human-readable text intended for use with standard spreadsheet programs to generate whatever analysis of test results is desired. Raw data is not stored in the global history file. Separate **raw data** files are saved for each unique array of raw data that is collected should there be a need to re-analyze results to adjust pass/fail limits as the number of SSP modules increases with time. A bit of vocabulary is required at this juncture to enable proper understanding. A hierarchical set of terms is presumed.

- There is a singular **program** that performs all **tests.**
- A **test** is a self-contained function of the **program** that exercises a given **parameter** of board functionality, or a *set* of related **parameters**, upon a given SSP module.
- Each **parameter** exercised in a given **test** may be classified as a *global* parameter or a *channel-based* parameter. Parameters are <u>not</u> raw data, but are calculated values that may be unambiguously tested upon a pass-fail basis against a min-max window of allowed values.
    - When there is only one instance of the parameter in a given SSP, such as the firmware revision can be read, or the average value of 1000 reads of a power supply voltage, this is considered a *global* parameter.
    - When there are multiple instances of a parameter in a given SSP, such as the slope in ADC counts per millivolt of the QI circuit response, these are considered *channel-based* parameters.
- A single-line **summary record**, unique per parameter per test, is generated in the global history file. The type of parameter (global or channel-based) is marked in the summary data, but all instances of a channel-based parameter are saved on a single line in the summary record.

    - Summary records will be different length lines but all fields will be separated with commas so that data will organize cleanly into columns one imported into a spreadsheet.
- For each **parameter** that is calculated from an array of multiple data values (e.g. average of 100 reads of something) the individual raw data values obtained from each unique calculation of the parameter are written to a separate humanly readable, comma separated, raw data file. Each raw data file shall contain two parts:
    - A series of lines whose values are identical to elements of the **summary record** in the global data file that identify module, date, test, parameter, etc.
    - The raw data header is followed by an arbitrary number of data point lines consisting of two values: the loop index of the measurement and the singular raw data value obtained for that loop index.

Format Details of Header information

The first data values written on each line of the **summary record** in the global data file consist of the *test header.* The exact same values of the *test header* are also written to the first few lines of each raw data file. This ensures that each raw data file may be correlated to each summary record in the global data file. The *test header* contains the basic information necessary to identify module, test, parameter, etc. and is shown here as a C structure.

```
struct test_header
   {
      int serial_number;    //entered serial number of DUT
      char run_time[50];    //string for date/time of test invocation
                            //in YYYYMMDD:HH:MM:SS format
      char test_name[50];   //string containing name of test
      char parameter[50];   //string containing name of parameter
      char username[50];    //entered name of user from main
      char user_comment[100];    //user comment as entered during setup
    };
```

Viewed as an Excel spreadsheet, the header portion of the global data file would look like this:

Serial #	Test Run Time	Test Name	Parameter	Username	Test-wide comment
3	20170805:14:22:37	Comm Test	Ping	Cundiff	Jackdaws love my big sphinx of quartz.
3	20170805:14:23:04	Comm Test	Read	Cundiff	Jackdaws love my big sphinx of quartz.
3	20170805:14:23:16	Comm Test	Write	Cundiff	Jackdaws love my big sphinx of quartz.
4	20170805:14:38:52	Comm Test	Ping	Cundiff	Now is the time for all good men to come to the aid of their party.
3	20170806:06:45:17	QI	Gain	Bill	Tune for minimum smoke.
3	20170806:06:52:01	QI	Stdev	Bill	Tune for minimum smoke.

Figure 3 – example of lines in a global data file showing only the header columns

Note that a given **test** may have different **parameters** that are tested, such as Ping, Read and Write for the Comm test.

A raw data file would be generated for each line in the global data file if there were any loops or collection of multiple data values. For instance, the QI test, Gain parameter shown in the 2nd to last line of the example global data file would generate a raw data file. The first few lines of that file would read as follows, matching the first few rows of the line in the global data file:

Serial #	3
Test Run Time	20170806:06:45:17
Test Name	QI
Parameter	Gain
Username	Bill
Test-wide com	Tune for minimum smoke.

Figure 4 - Example of header lines in a raw data file

Test result data in the global data file

In the global data file, following the *test header,* each test shall issue a test-specific series of *test results.* These test results follow after the header, with all results from each individual run on the same line as the header. Viewed as a C structure, the various elements comprising a test result are shown.

```
struct test_result
   {
     char channel[10];     //channel number or "GLOBAL" if not
                           //channel-based
     double test_result;   //numerical result
     double win_min;       //minimum value of result to pass
     double win_max;       //maximum value of result to pass
     char status[15];      //"PASS" or "FAIL" or "ERROR"
    };
```

The single **test header** and however many **test result(s)** are written in 'append' mode as comma-separated values to the global output file. One line will be generated for each unique parameter of each test. If a given parameter of a given test is measured across multiple channels, a single line in the file shall contain the measurement results obtained for every channel. An example of how this would look once imported into Excel is shown as Figure 5.

Serial #	Test Run Time	Test Name	Parameter	Username	Test-wide comment	Channel	Result	Win Min	Win Max	STATUS
3	20170805:14:22:37	Comm Test	Ping	Cundiff	Jackdaws love my big sphinx of quartz.	GLOBAL	1	1	1	PASS
3	20170805:14:23:04	Comm Test	Read	Cundiff	Jackdaws love my big sphinx of quartz.	GLOBAL	1	1	1	PASS
3	20170805:14:23:16	Comm Test	Write	Cundiff	Jackdaws love my big sphinx of quartz.	GLOBAL	1	1	1	PASS
4	20170805:14:38:52	Comm Test	Ping	Cundiff	Now is the time for all good men to come to the aid of their party.	GLOBAL	0	1	1	FAIL
3	20170806:06:45:17	QI	Gain	Bill	Tune for minimum smoke.	0	17.3784	16.5	17.1	FAIL
3	20170806:06:52:01	QI	Stdev	Bill	Tune for minimum smoke.	0	0.1	0.02	0.12	FAIL

Test Run Time	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS
20170805:14:22:37															
20170805:14:23:04															
20170805:14:23:16															
20170805:14:38:52															
20170806:06:45:17	1	17.05	16.5	17.1	PASS	2	16.3	16.5	17.1	FAIL	3	16.732	16.5	17.1	PASS
20170806:06:52:01	1	0.02	0.02	0.12	PASS	2	0.01	0.02	0.12	FAIL	3	0.11	0.02	0.12	PASS

Test Run Time	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS
20170805:14:22:37																				
20170805:14:23:04																				
20170805:14:23:16																				
20170805:14:38:52																				
20170806:06:45:17	4	17.3784	16.5	17.1	FAIL	5	17.3784	16.5	17.1	FAIL	6	17.3784	16.5	17.1	FAIL	7	17.3784	16.5	17.1	FAIL
20170806:06:52:01	4	0.15	0.02	0.12	FAIL	5	0.16	0.02	0.12	FAIL	6	0.16	0.02	0.12	FAIL	7	0.16	0.02	0.12	FAIL

Test Run Time	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS
20170805:14:22:37																				
20170805:14:23:04																				
20170805:14:23:16																				
20170805:14:38:52																				
20170806:06:45:17	8	17.3784	16.5	17.1	FAIL	9	17.3784	16.5	17.1	FAIL	10	17.3784	16.5	17.1	FAIL	11	17.3784	16.5	17.1	FAIL
20170806:06:52:01	8	0.16	0.02	0.12	FAIL	9	0.16	0.02	0.12	FAIL	10	0.16	0.02	0.12	FAIL	11	0.16	0.02	0.12	FAIL

Figure 5 - full-width lines in global data file including all summary test results

Because there are many channels in an SSP, the global data file of necessity has many columns. Blowing up the left side of Figure 5, we'll zoom in on the relevant per-channel information. The first six columns of Figure 5 have already been shown as Figure 3 – these are the test header columns intended to be used for sorting purposes. Following the test header columns there will be one or more 5-column wide *channel result groups* as shown in Figure 6.

Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS	Channel	Result	Win Min	Win Max	STATUS
GLOBAL	1	1	1	PASS										
GLOBAL	1	1	1	PASS										
GLOBAL	1	1	1	PASS										
GLOBAL	0	1	1	FAIL										
0	17.3784	16.5	17.1	FAIL	1	17.05	16.5	17.1	PASS	2	16.3	16.5	17.1	FAIL
0	0.1	0.02	0.12	FAIL	1	0.02	0.02	0.12	PASS	2	0.01	0.02	0.12	FAIL

Figure 6 - Zoomed-in view of columns G-U of Figure 5

The first four entries shown in Figure 3 were *global* tests, in the sense that communication test parameters such as "did it ping", "can I read" or "can I write" have only one occurrence for each SSP (at least until we're required to support multiple ports within the Ethernet interface). However, the example QI/gain and QI/Stdev parameters exist once per channel in the SSP, so there is a 5-column block for each channel. Each *channel result group* contains the channel number, the summary result used for the pass/fail criterion of the parameter, the min/max range allowed for the summary result and the pass/fail result decreed by the program. It is not expected that different min/max criteria per channel will be implemented at this time, but the format allows for such to be specified if needed in the future (for example, edge channels 0 and 11 might be allowed slightly different mean or Stdev characteristics for pedestal than interior channels like 5 or 6).

The global data file is intended to allow higher-level statistics across many runs of a given test over multiple modules to be generated so that production variances and/or aging can be extracted. As previously stated, the global data file does not contain any raw data; separate raw data files are saved.

Raw Data File format

As shown previously in Figure 4, each raw data file will contain identical information to columns A-F of each line of the global data file to allow correlation between the global pass/fail results and the individual raw data files. Raw data files shall follow a general format in which the header is followed by an array of all the raw data, exemplified by Figure 7.

Serial #	3												
Test Run Time	20170806:06:45:17												
Test Name	QI												
Parameter	Gain												
Username	Bill												
Test-wide com	Tune for minimum smoke.												
Loop Index	Vset	Channel 0	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7	Channel 8	Channel 9	Channel 1	Channel 1
1	0	1	0	0	0	1	0	1	1	0	0	1	0
2	0	1	1	1	0	1	0	0	1	1	0	0	0
3	0	1	1	1	1	1	0	1	1	0	1	0	1
4	0	1	1	1	0	0	1	1	0	1	0	1	1
5	0	0	1	0	1	0	1	1	0	0	0	1	1
6	0.1	18.4	18.4	17.4	18.4	18.4	18.4	18.4	18.4	18.4	17.4	17.4	18.4
7	0.1	18.4	17.4	18.4	18.4	18.4	17.4	18.4	18.4	18.4	17.4	18.4	17.4
8	0.1	17.4	18.4	18.4	17.4	18.4	17.4	18.4	17.4	17.4	17.4	17.4	17.4
9	0.1	18.4	18.4	17.4	17.4	18.4	17.4	17.4	17.4	18.4	17.4	18.4	18.4
10	0.1	17.4	18.4	18.4	18.4	17.4	18.4	18.4	17.4	17.4	17.4	18.4	17.4
11	0.2	35.8	35.8	34.8	35.8	34.8	34.8	34.8	34.8	34.8	34.8	34.8	34.8
12	0.2	35.8	35.8	35.8	35.8	35.8	35.8	34.8	35.8	35.8	35.8	35.8	34.8
13	0.2	35.8	34.8	35.8	34.8	34.8	34.8	34.8	35.8	35.8	35.8	34.8	35.8
14	0.2	35.8	34.8	35.8	35.8	35.8	34.8	34.8	35.8	34.8	34.8	34.8	35.8
15	0.2	34.8	34.8	34.8	34.8	35.8	34.8	34.8	34.8	35.8	35.8	34.8	34.8
16	0.3	53.2	52.2	52.2	53.2	53.2	53.2	52.2	52.2	53.2	52.2	52.2	53.2
17	0.3	52.2	53.2	52.2	52.2	52.2	52.2	53.2	53.2	53.2	52.2	52.2	52.2
18	0.3	52.2	52.2	53.2	53.2	52.2	52.2	52.2	52.2	52.2	52.2	53.2	53.2
19	0.3	53.2	53.2	52.2	52.2	53.2	53.2	53.2	52.2	52.2	53.2	53.2	52.2
20	0.3	53.2	53.2	53.2	53.2	52.2	53.2	53.2	53.2	53.2	52.2	52.2	53.2
21	0.4	70.6	70.6	69.6	69.6	70.6	70.6	70.6	69.6	69.6	69.6	69.6	69.6
22	0.4	70.6	69.6	69.6	70.6	70.6	69.6	69.6	70.6	69.6	70.6	70.6	70.6
23	0.4	69.6	70.6	70.6	69.6	69.6	70.6	69.6	70.6	69.6	69.6	70.6	69.6
24	0.4	70.6	69.6	70.6	70.6	70.6	70.6	69.6	70.6	69.6	69.6	70.6	69.6
25	0.4	70.6	70.6	69.6	69.6	70.6	69.6	70.6	69.6	69.6	70.6	70.6	69.6

Figure 7 - typical format of a raw data file

The general expectation applied to raw data files is that each file shall contain all the readings from each channel for each setting of a given stimulus, and that multiple readings shall be taken for each unique stimulus. While the example in Figure 7 shows only a single stimulus value "Vset", the raw data files will be allowed the flexibility to have any arbitrary number of columns to describe all stimuli associated with a given reading and what each column of numbers means. The software shall write as much descriptive information as required in as many rows as required following the header. In the example of Figure 7, a single descriptive line has been written, highlighted in red.

General notes regarding analysis of data

The purpose of the comma-separated text format is to facilitate import of the data file into Microsoft Excel or other spreadsheet program. The one-line-per-parameter-per-test format of the global data file allows sorting by many different combinations of columns to generate a variety of production trend charts. By contrast, the raw data file format is not intended for sorting but instead for the experienced user of Excel or Root to perform whatever detailed analysis is desired of a specific test's dataset.

Storage of pass/fail criteria used in tests

All pass/fail criteria used in tests will be stored in a single file. As it is undesirable to use a simple text file that anyone could modify at any time, the pass/fail criteria will be stored in a C *include file* (AKA ".h file") as a data structure. Include files are simple text and easily read by a human, but for any change to take effect the program must be recompiled.

General notes on quality control

The data format used for the global data file records not only the entire test history of each module but also records the pass/fail criteria used for each execution of each test. Thus not only can the entire history of a given module be retrieved but also any changes applied to the testing procedure over time are logged. The use of Box Sync along with a single global data file provides a mechanism by which multiple computers in multiple places may use the same test software, but still aggregate all test records into a single place.