CFD Testing Manual

by Thomas Roche

Table of Contents

7 Synopsis

The Cross Fractional Discriminator (CFD) acts as the level one trigger in the VERITAS telescope array. Its purpose is to determine whether or not the photon(s) incident on a single PMT in one of the telescope cameras resulted in a large enough voltage to be counted as an event. Although each CFD is schematically the same the various components can differ slightly. As a result each board must be tested and calibrated to ensure (barring later failure) accurate and predictable operation. This document will instruct the reader in all of the necessary aspects of the testing regime.

8 Equipment

The computer aptly named cfd.astro.ucla.edu is the central hub for all of the testing procedures done on the CFDs. Using a network of GPIB and SERIAL connections most of the test equipment can be controlled via this computer. A tally of the equipment needed to run all of the tests is as follows: an oscilloscope, signal generator, NIM crate, Night Sky Background (NSB) Generator and a CFD Test Rig. Fortunately there are 15 test rigs which speeds up the process considerably.

8.1 Computer controlled equipment

This section will let you know how to communicate with the various devices and give you the some of the specific information you will need to get started using them.

8.1.1 GPIB

8.1.1.1 Oscilloscope

The oscilloscope used is a Tektronix TDS 3054B. It has a bandwidth of 500 MHz and is capable of taking 5 billion samples per second. It communicates with the computer via a GPIB interface. The functions of the scope are too many to list here but it is suggested that the reader familiarize himself with the device. The primary use for the the scope with regard to CFD testing is measuring output pulse width and an overall output check. The width of the output pulse is measured within 0.25 ns in the width test and output ranges are checked in the mode test.

8.1.1.2 Signal Generators

8.1.1.2.1 SRS DG535

The Stanford Research Systems generator is used to create output pulses that can simulate photon showers or simply provide a consistent trigger for the CFD. It has four output channels. The device can be triggered externally but the internal trigger works fine as well for frequencies up to 1 MHz. Each channel's output can be delayed in reference to any other channel or the trigger. This comes in handy when measuring jitter and width. One channel's output can be sent to the scope to act as a reference. The output of the CFD may then be sent to another channel on the scope. By setting the delay on the scope triggering pulse you can find the output of the CFD and make measurements.

This unit also has a GPIB interface. It is possible to use LabView to program the values for all of the functions. It is important to familiarize yourself with all of the menus though. It is also vital to have a good understanding of the output pulses produced by this device. There seems to be a problem in the couple of nanosecond range. The output has a short overshoot of the desired voltage. This can make sending a pulse of a specific size and shape difficult, but not impossible. By taking a reading of the pulse, that actually reaches the CFD while adjusting the settings, the device's behavior can be easily understood.

8.1.1.2.2 TEGAM AWG

This may need documentation but for now all I know is that it is an arbitrary waveform generator that can produce signals at 100 megasamples per second.

8.1.2 Serial

8.1.2.1 MOXA

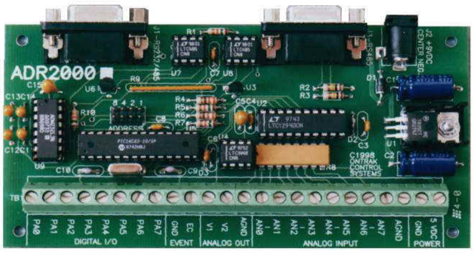
The CFD computer is equipped with two multi-port PCI serial cards.

They are made by the company MOXA. Each of the cards has an octopus cable attached to it which has 8 serial connections. Linux numbers the added serial interfaces as follows:
Card 1: /dev/ttyA11 - /dev/ttyA18
Card 2: /dev/ttyB11 - /dev/ttyB18
In LabView the interfaces are called MOXA1 – MOXA15. The tty device that corresponds to the specific device is sequential. TtyA11 = MOXA1 and ttyB17=MOXA15. You can connect to a port directly by typing:

minicom moxa#

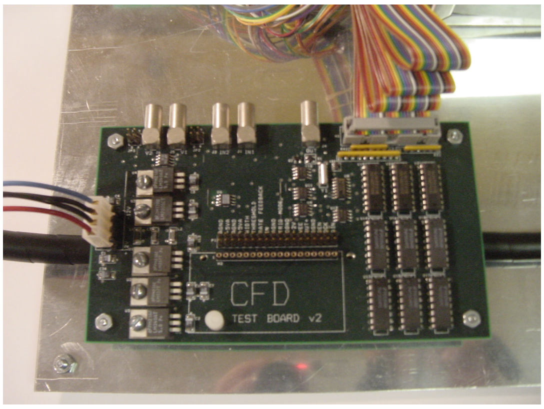
At the command line. Where # is the number of the moxa device you want to connect to.

8.1.2.2 ADR2000



The ADR2000 handles all of the communication between the computer and the CFD Test Board. It is used to program the CFD and read the currently programmed values. It is also used to interpret the output of the scalar and oscillating crystal on the test board. When minicom is used to connect to a particular moxa port you are actually opening a direct serial connection to the ADR2000. The most difficult data acquisition done by the ADR2000 is reading data from the scalar. To accomplish this one bit must be read at a time and gathered in to a 24-bit binary string. This is done in the program *DACBthreshold* which is discussed below. For more information on the specific commands that are used to read data and how to interpret it please refer to the ADR2000 manual.

8.1.2.3 CFD Test Board



The CFD Test Board (CTB) is the piece of hardware that the actual CFD plugs in to. Its goal is to simulate the front end channel of the flash ADC boards used in the telescope with some extra modifications. It has two inputs so that pulse signals can be mixed. There is a ECL to NIM converter that translates the output of the CFD in to the 50 ohm NIM standard. This is done reasonably well, but be aware that there is an overshoot in the output pulse. There are also two additional outputs that can be programmed by the ADR2000 to output a set voltage. This is used to power an LED in the Night Sky Background (NSB) generator. There is also a scalar and a crystal oscillator which is used to measure triggering rates.

8.1.3 NSB Generator

The Night Sky Background Generator simulates the conditions at the site. It is used to inject noise in to a signal that is similar to the noise present at the telescope. It is housed inside of a 40cm cylindrical tube that has been closed off on each end. Inside there is a Photonics PMT and amplifier. One end has an LED with LIMO connection the other has all of the leads necessary to power the PMT and preamp, output, input and current monitor.

8.1.3.1 Connections

The front plane of the NSBG has 7 connectors:

1) High Voltage: Attach the Red Cable from the High Voltage power source on the rack
2) Current Monitor: Attach the cable from the ADR2000 on MOXA9 to this input. It monitors the current coming from the PMT which allows the user to calculate the rate of photoelectrons per nanosecond.
3) +5 Volts: This is the Red Banana Plug connection. Connect the Red Cable with Red End from the power supply on the rack.
4) -5 Volts: This is the White Banana Plug connection. Connect the Blue Cable with Red End from the power supply on the rack.
5) Ground: This is the Black Banana Plug connection. Connect the Black Cable with Black End from the power supply on the rack.
6) Output: This is the BNC without the black washer behind it. This

output should be connected to one end of the 125' 75 Ohm Cable sitting on top of the rack. The other end of the cable should go to input number 1 on the MOXA9 Test Rig.

7) Charge Injection: This is the BNC with the black washer behind it. Connect the signal from the signal generator to this input. That may come from the Linear Fan-Out or direction from the generator. This signal is added to any signal coming out of the PMT.

8.1.3.2 PMT

The Photonics PMT sitting inside of the NSBG is very sensitive and can detect a single photoelectron (pe). It was calibrated using the vi pmtcalibration.vi.

8.1.3.2.1 HV Power Supply

This power supply can be left alone for the most part. Once the PMT is calibrated the desired voltage should not be changed unless it needs to be recalibrated for some reason. The high voltage is currently set at:

8.1.3.2.2 LED

A small Light Emitting Diode is placed inside the NSBG in front of the PMT. The voltage across the diode can be altered via the connection to the MOXA9 test rig. It is regulated by the ADR2000 board. Adjusting the voltage changes the rate of photo emission and , in turn, affects the current coming out of the PMT.

8.1.3.3 Preamplifier

This device amplifies the signal from the PMT. It also mixes the signal from the charge injection input with the PMT signal.

8.1.3.3.1 Power Supply

8.2 NIM

8.2.1 Linear Fan-Out

8.3 *Laboratory Layout*

9 Testing

This next section will give a through description of all of the tests that must be performed on each CFD for proper calibration and performance evaluation. The discussion is meant to be theoretical in nature and not specific to the programs used to gather the intended results. Once an explanation of why and what has been given, how will be presented in section 6 (Software).

9.1 List of tests with minimal descriptions

Mode Test:	Quick overall check.
DAC Linearity Test:	Check linearity of both DACs on CFD.
Width Test:	Measure/Calibrate width of output pulse.
RFB Linearity Test:	Determine behavior of RFB circuit.
Noise Calibration Test:	Create noise curves.
DACB Threshold Test:	Measure threshold accuracy and efficiency.
RFB Test:	Measure effect of noise on RFB circuit.
Timing Test:	Determine jitter, efficiency and mean of CFD mode.

9.2 Mode Test

The mode test is the very first thing (after labeling) that should be done for each CFD. The idea is to quickly check each of the functions and make sure the easily measured values look right. Threshold and width are changed to their extremes, both trigger modes are checked for operation and rate feed back is altered. The test itself amounts to reading voltages and looking at the output on the oscilloscope and finally recording a result of pass/fail in a database.

9.3 Coefficient Determining Tests

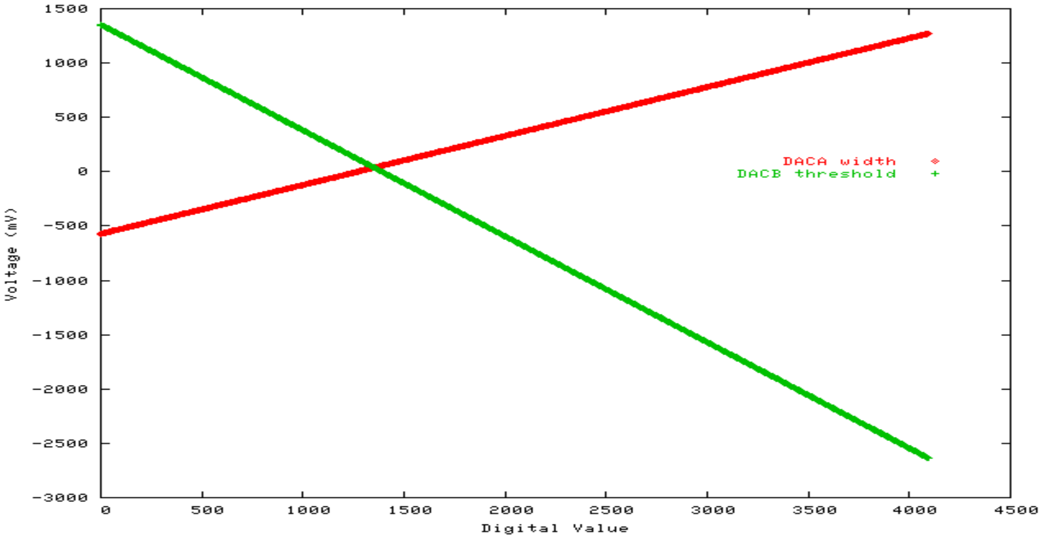
9.3.1 Purpose

The following tests' primary purpose is to provide data about each CFD that will be used to program them during the run time of the telescopes. Width, threshold and rate feedback must calibrated for every CFD. These calibrations result in coefficients for curves that give the digital values that should be programmed for desired physical values. There is also a secondary purpose for all of these tests. Not only do they provide calibration data but they can also point out failures that the mode test did not recognize. In fact, this secondary purpose is the first thing looked for after a test is performed.

9.3.2 Digital to Analog Converter (DAC) Linearity Tests

The DAC linearity tests are the most exhaustive of all of the tests. There are two DACs in on the CFD. One controls width and the other threshold.

Each has 4096 possible settings or channels. Each channel must be checked for correct operation. To be confident, one hundred measurements are taken at each of these 4096 setting for each DAC. This amounts to over 800,000 points of data for each board. In a normal board the data taken from this test should lie along an extremely straight line. Below is an example of the kind of data produced that should be produced by a good board.



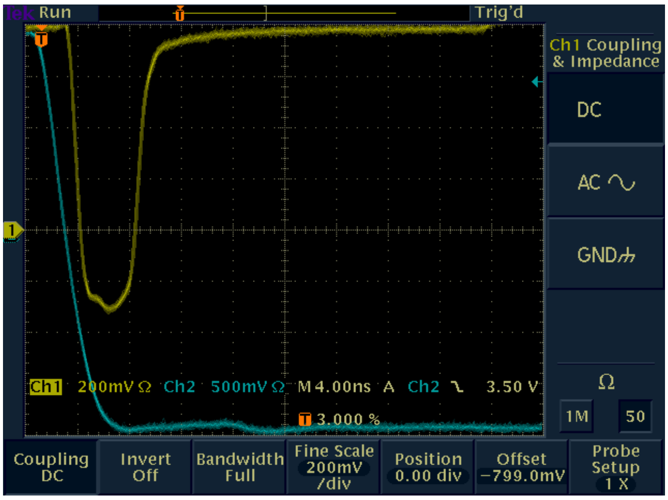
Although it may not appear so, these lines are comprised of thousands of individual points. Any variation in this level of linearity is indicative of a malfunctioning CFD. However, these curves tend to shift around and where it lies is where the calibration comes in. These curves can easily be fit to an equation of the form:

y = m*x +b

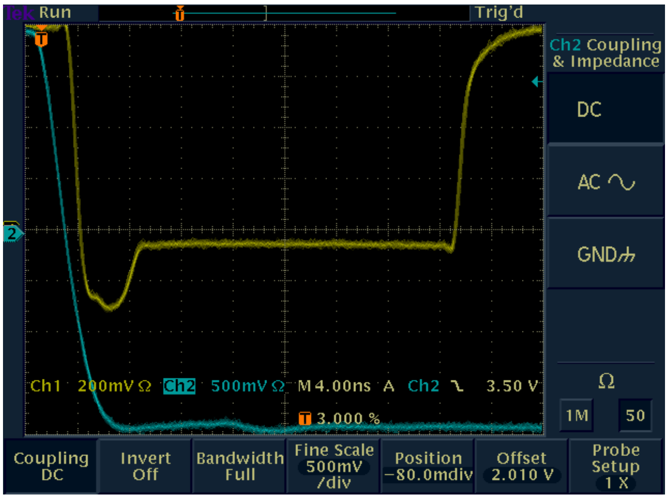
It is only necessary to fit the threshold curve though. The voltage of DAC B is the actual voltage used by the discriminator.

9.3.3 Width Test

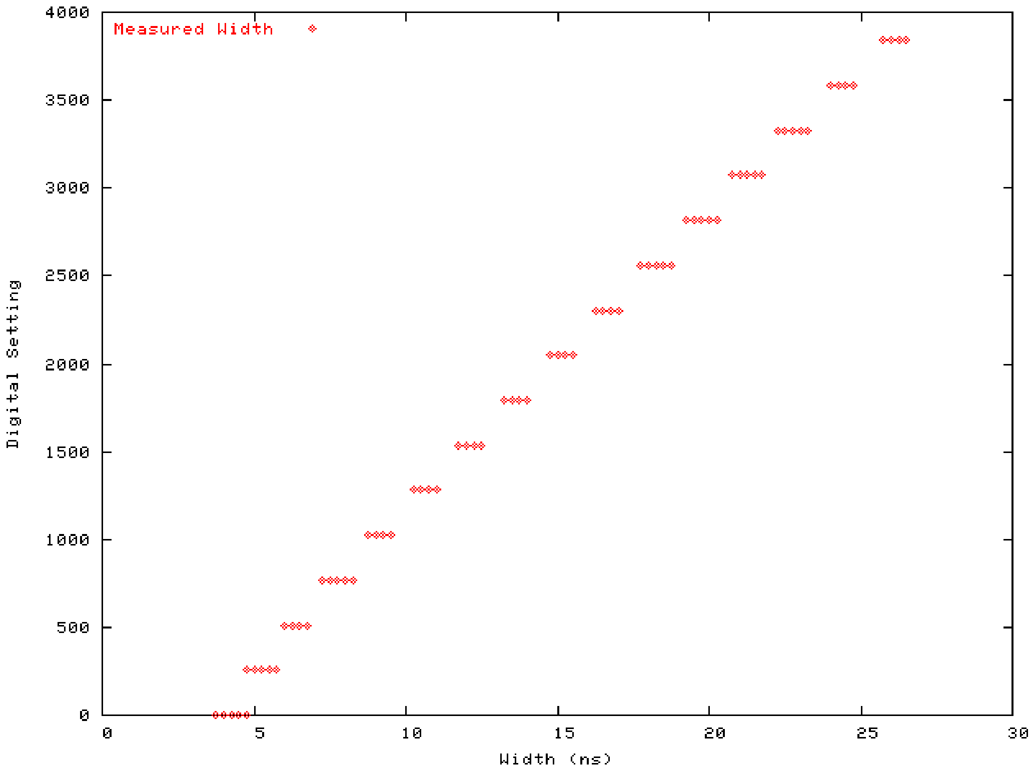
The width DAC voltage is not the whole story. This width voltage is translated in to an output pulse length. A capacitor is discharged. While discharging the output from the CFD remains constant. The length of this discharge determines the width of the pulse. The voltage emerging from DAC A determines how long the discharge is. However, not all of the capacitors are exactly the same so the output width varies as the DAC and as the internal properties of the circuit involved. Therefore the actual width of the output has to be measured. It looks like this:



The yellow trace is the output of the CFD. The blue trace is the reference line. It is a pulse generated at the same time as the pulse that triggers the CFD. Although this is not very important at this point of the testing. Later when we measure jitter in the timing test it is essential to have this kind of a reference. In this case DAC A is programmed with a value of 0. This results in the narrowest possible pulse. This next image is the result of programming 4095 for DAC A ... The widest possible setting:



Data gathered by this test should look something like this:



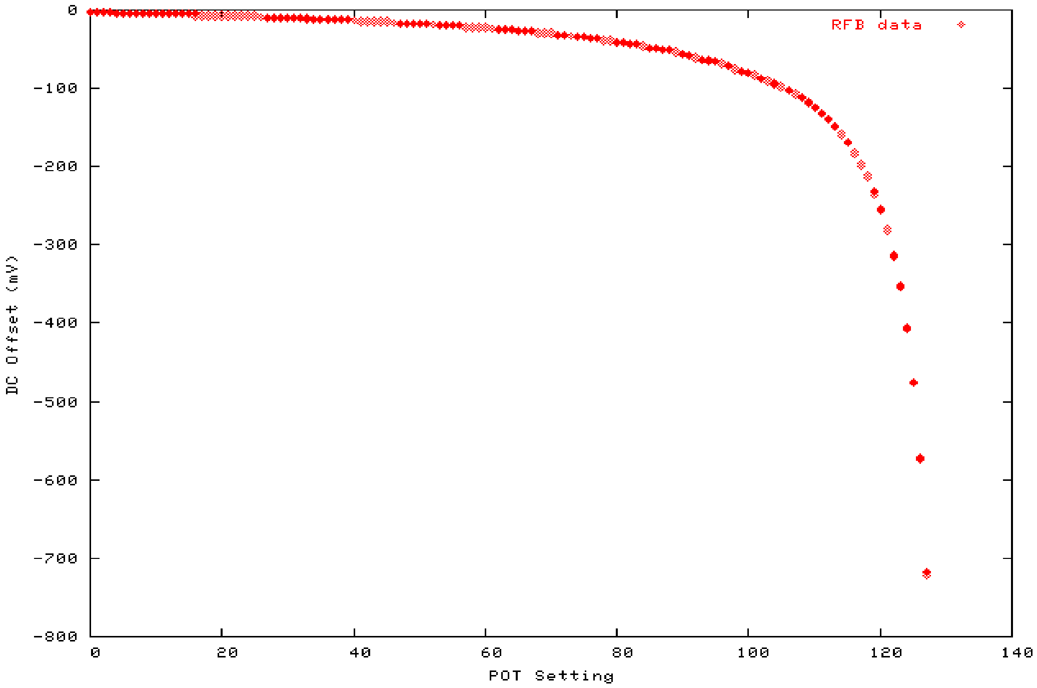
Each of these points represents a point on a histogram. The spread is not as wide as it seems. Most of the measurements fall into the middle of each group. The outside points represent one or two samples. The line that fits this curve is a bit more involved:

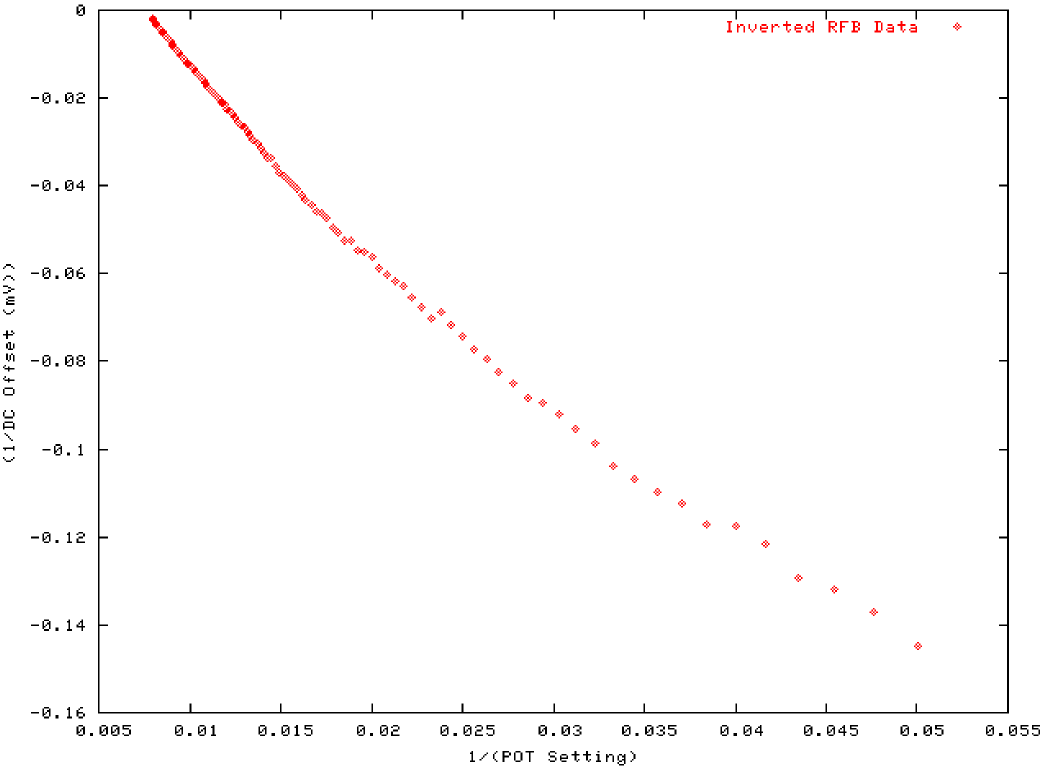
$y = A*(x/x_0 -1)^\wedge(1/3) + B*(x/x_0 -1)$

Where x_0 is the average width with DAC A set to zero. Coefficients A and B are needed to determine the programmed value during runtime.

9.3.4 Rate Feed Back (RFB) Linearity Test

Rate feed back is an vital part of the level one trigger. It's purpose is to suppress the threshold when the frequency of triggering starts to rise. It is controlled by a variable resistor (potentiometer). The RFB circuit responds by changing the threshold voltage in units of mV/MHz. So, the higher the frequency of triggers the larger the DC offset. In this test a consistent pulse with a frequency of 1 MHz is delivered to the CFD. The DC offset should be measured 100 times at each of the 128 potentiometer settings. The reciprocal of the results fit a straight line. Below are pictures of the actual data and the same data inverted.


RFB data
DC Offset (mV)
POT Setting
0
-100
-200
-300
-400
-500
-600
-700
-800
0
20
40
60
80
100
120
140


Inverted RFB Data
(1/DC Offset (mV))
1/(POT Setting)
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
-0.16
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
0.055

These lines are not as perfect as the DAC plots but the interesting region is as the inverted plot approaches zero. The variation in the line at smaller DC offsets is a result of non ideal pulse (there is some ringing in the high frequency range of the pulse).

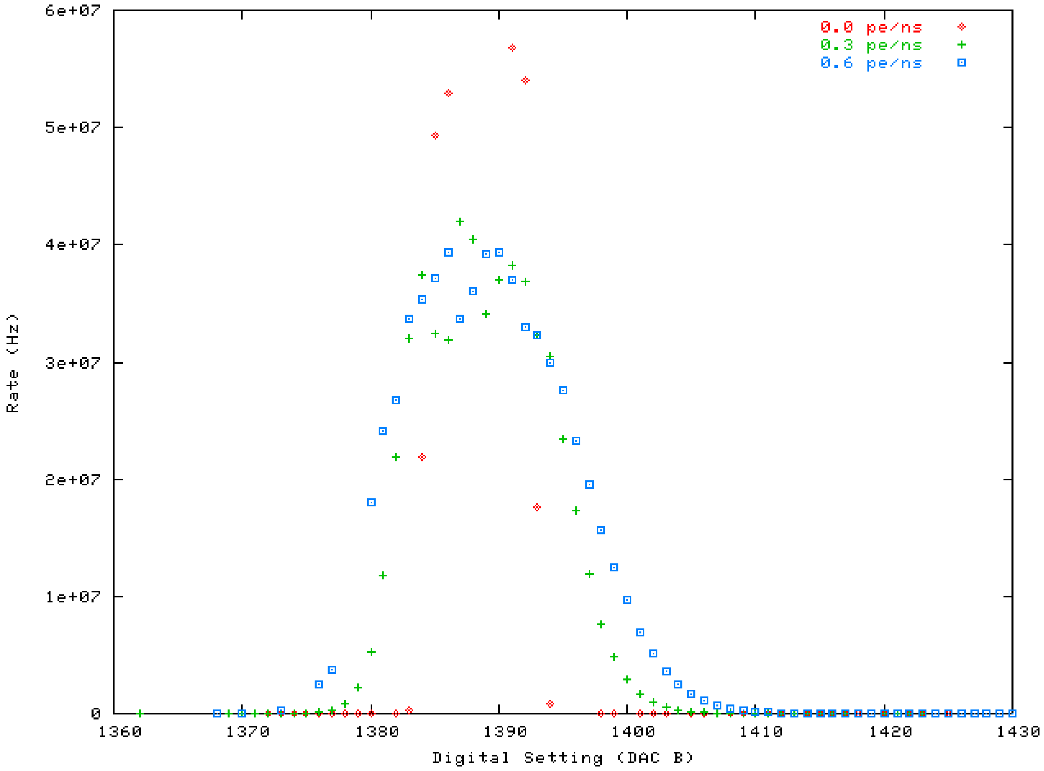
9.4 Fine Tuning Tests

9.4.1 Purpose

The rest of the tests provide important information about the behavior of the CFDs under specific conditions. The results will eventually be used to aid in the off line analysis of the runtime data. After a CFD has gone through and passed the tests listed above one can be fairly confident that they are properly functioning boards. The rest of the data gathered acts as a fine toothed comb, separating the data more precisely since certain behaviors of each board can be accounted for.

9.4.2 Noise Calibration Test

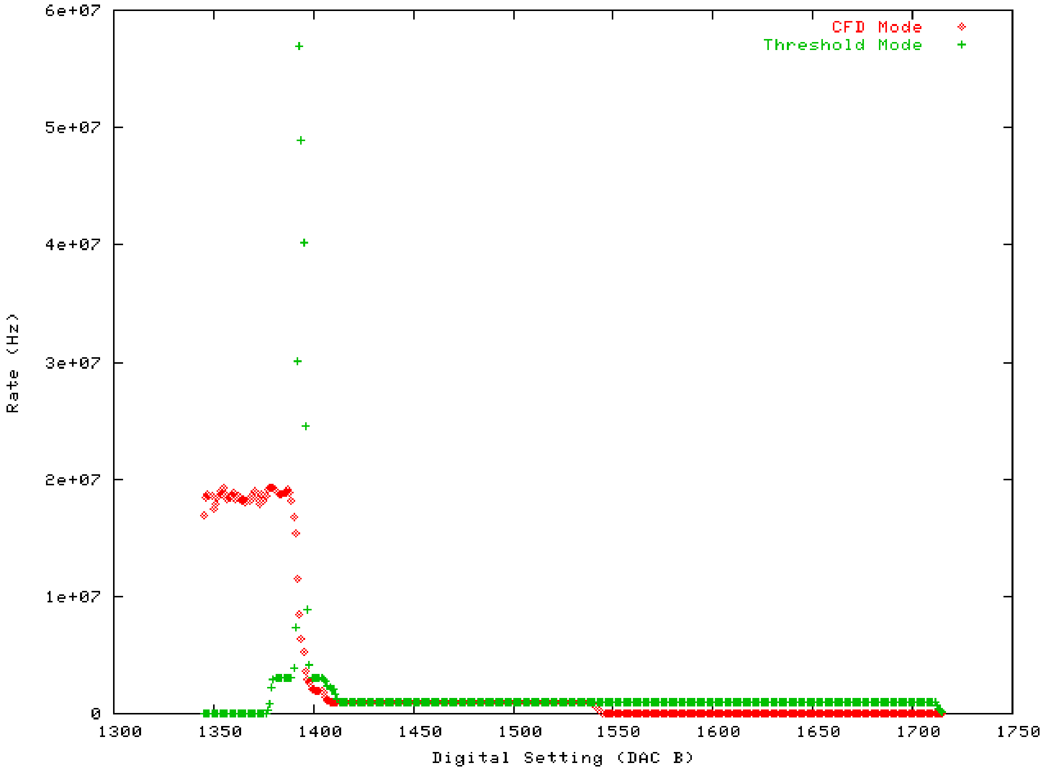
The most important discoveries in this experiment will most likely come at energies very close to the noise level. Therefore, it is of utmost importance to understand and be able to identify the characteristics of noise in the system. This test is not worried about noisy electronics but rather the night sky background radiation. As described above the NSB generator can simulate a controlled version of it. This test provides data that will give traces of how CFDs respond to noise at a predetermined rate. Both modes should be tested at at least three different noise levels. The following image shows how a particular CFD behaves in normal threshold mode with 0.0 pe/ns, 0.3 pe/ns and 0.6 pe/ns noise injected.



This test can also be used to show that the measured threshold voltage agrees with actual triggering rates as the peak of this graph should be located at the 0 volts.

9.4.3 DACB Threshold Test

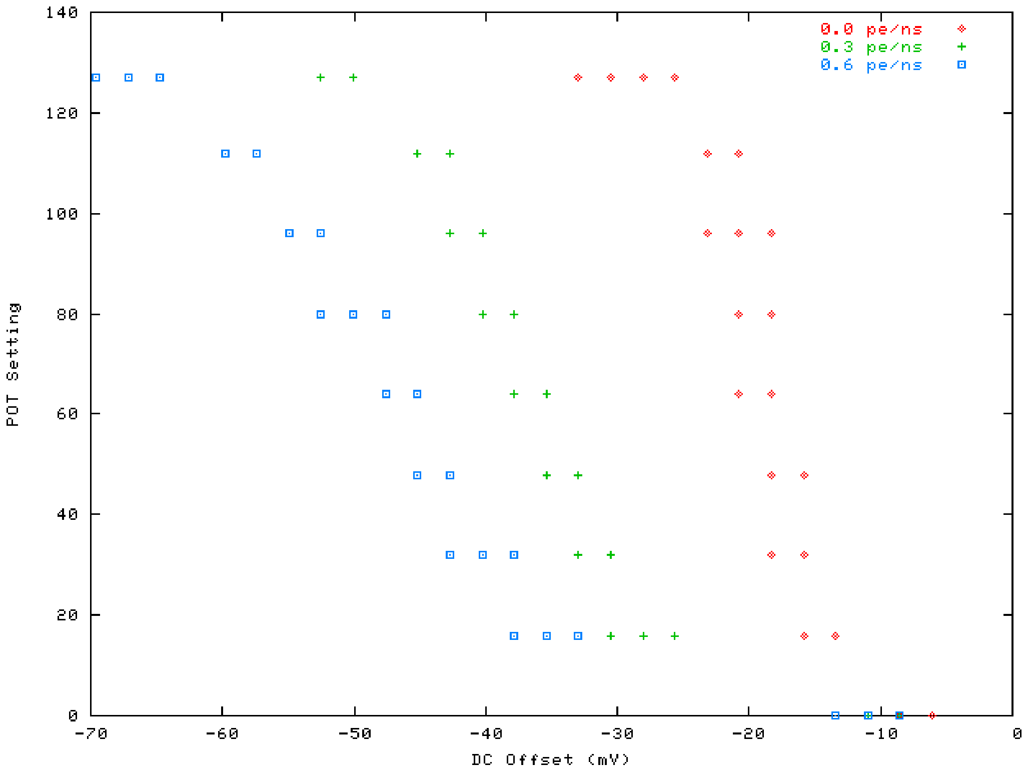
This test measures many things. The idea is to inject a fairly large pulse with a large amplitude (1 MHz in this case) and measuring the trigger rate while iterating through the threshold settings. Knowing the size and shape of the pulse and using it consistently allows this data to be used to determine how close the threshold can be to the size of the pulse and for it to still trigger. This test should measure performance for both CFD and threshold modes. The third batch of CFDs has been modified to use a different fraction (to 0.3) which will greatly improve the efficiency close to the threshold but the data shown below was taken with a CFD using 0.7 fraction. The effect is obvious. Additionally this test will give another check for the zero of the threshold as there is a rate spike at 0 in threshold mode.



The figure above shows how the CFD triggering rate drops much earlier (about 150mV before) than the threshold mode. Look for improvements in the new batch.

9.4.4 RFB Test (With Noise)

This test is pretty self explanatory. A variety of noise levels are directed to the input of the CFD and the RFB setting is varied. The resulting DC offset is measured 100 times for each setting. It is expected that more noise will result in a larger (in magnitude. More negative actually) DC offset and the following sample set of data verifies that assumption.



This data will be used to determine what the optimal RFB setting is based on noise statistics at the site.

9.4.5 Timing Test

The timing test incorporates all of the functionality of the CFD. A few things are calibrated. Jitter, efficiency and mean arrival times of triggers are all calculated from the data gathered. Noise at different levels is injected into the system in addition to a regular photon shower like pulse. The idea is to simulate the telescope conditions as much as possible so that confidence levels will be able to be predicted. The oscilloscope should be used to record output waveforms and their position relative to a control pulse.

10 Data

The CFD data is located in two places. At first this was because LabView is unable to output to a mySQL database. So the programs in LabView would output a file with raw data in a particular format and a program would later put that data into the database. But upon further thought it was realized that this method has more than simple practical benefits. The raw data files are not kept on the same computer as the database. The database is backed up but the backups remain on the computer with the database on it. If by some unfortunate event one of the two computers caught on fire and disintegrated into tiny bits the data would survive. This kind of redundancy is nice when the time and man power required to gather the data is considered.

10.1 Data Files and Locations

The data files (and all things pertaining to the CFDs) are located on cfd.astro.ucla.edu in the cfduser's home directory:

/home/cfduser/CFDdata/CFD-*/

There is a separate directory for each of the tests:

/home/cfduser/CFD-DACBthreshold/
/home/cfduser/CFD-DAClinearity/
/home/cfduser/CFD-NoiseCalibration/
/home/cfduser/CFD-RateFeedBack/
/home/cfduser/CFD-RFBlinearity/
/home/cfduser/CFD-Timing/
/home/cfduser/CFD-width/

The data for each test performed on each CFD is stored in a separate file. The files are uniquely identified by the CFDid of the CFD. A sample file name for a DACAlinearity test is as follows:

/home/cfduser/CFD-DAClinearity/0387DACAlinearity

This identifies the CFD and the test. Inside of this file the data looks like this:

```
0 -571.289062 33
0 -568.847656 66
1 -571.289062 15
1 -568.847656 80
1 -566.406250 3
2 -571.289062 3
2 -568.847656 63
2 -566.406250 33
3 -568.847656 54
3 -566.406250 43
4 -568.847656 40
4 -566.406250 58
```

This is the binned histogram data for a test. Once the data in a file has been put in to the database the file is renamed like this:

/home/cfduser/CFD-DAClinearity/0387DACAlinearity.inserted

This avoids duplicate entries in the database.

10.2 Database

10.2.1 Location

The grand collection of the data is housed in a mySQL database on the machine *gamma1.astro.ucla.edu*. It can be accessed from any machine in the network provided the proper permissions have been granted. To access the database type the following from a terminal on cfd:

mysql -h gamma1 cfd

This will connect you to the server.

10.2.2 CFD Database Structure

The structure of the CFD database is very similar to the directory structure on cfd.astro.ucla.edu. The nomenclature of tables mimics the VERITAS database at the experiment site. Below is a brief output that shows a connection to the database and a list of the tables inside of it. Lines with user input have been highlighted for your convenience.

```
[troche@cfd CFDManual]$ mysql -h gamma1.astro.ucla.edu cfd
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 990 to server version: 4.1.9-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show tables;
+-------------------------+
| Tables_in_cfd           |
+-------------------------+
| tblCFD_DACAlinearity    |
| tblCFD_DACAwidth        |
| tblCFD_DACBlinearity    |
| tblCFD_DACBthreshold    |
| tblCFD_DACCoefficients  |
| tblCFD_Info             |
| tblCFD_NoiseCalibration |
| tblCFD_RFBlinearity     |
| tblCFD_RateFeedBack     |
| tblCFD_Timing           |
+-------------------------+
10 rows in set (0.00 sec)

mysql>
```

There are two tables here that do not coincide with directories on the cfd machine. Info and DACCoefficients. The info tables keeps a record of tests that have been performed on a CFD and whether or not it passed or failed. It also has a comments field so that problems with particular CFDs can be documented. The coefficients table has the coefficients that are used by the telescope computers to program the CFDs on site during runtime. The data in that table is copied to the database on site.

11 Software

Plenty of software has been developed to do all of this testing. The bulk of it is written in either LabView or C++. There are a few perl scripts in the mix as well. Everything is on the cfd.astro.ucla.edu machine except for the program at the site on the vdaq machine that propagates the coefficients to the VERITAS database.

11.1 LabView

LabVIEW
Graphical Programming for Instrumentation

LabView is a graphical programming language that specializes in communicating with external test equipment like signal generators and oscilloscopes. The programs are called VIs. A VI may contain many sub VIs which function like subroutines. The subroutines are in fact VIs as well. They have input and output parameters like a function in a traditional programming language.
LabView must be started before VIs can be opened. The binary is:

/usr/local/bin/labview

All of the VIs are located in:

/usr/local/lv70/user.lib/

11.1.1 Data Gathering Tests

LabView is used for five (5) of the calibration tests: Mode, DACAwidth, NoiseCalibration, RateFeedBack and Timing. A detailed how-to contains information on performing these tests.

The Mode test isn't like the rest of the tests. You simply use the CFD.vi program to adjust the values programmed on the CFD. After that you use the command line program "modetest" to make an entry in the database that specifies that the test has been performed and whether or not it passed.

The files for the rest of the tests are named:

DACAwidth.vi
NoiseCalibrationTest.vi
RateFeedBack.vi
RFBTiming.vi

The user should familiarize himself with the operation of these programs. If he has a good understanding of the test equipment they should not pose much difficulty as the most intricate parts are emulations of the scope, signal generator and NSB generator.

11.1.2 Diagnostic Programs

There are some important diagnostic programs in LabView.

CFD.vi
This is used for the Mode test but it is also used any time you would like to quickly communicate with he CFD. It is a complete interface with everything needed to monitor the full range of a CFD's operation. You can change the mode, set the width and threshold, adjust rate feed back and read the currently set values.

crystal_frequency_counter_big.vi
This program is used to measure the frequency of the crystal oscillating on the CTB. Unfortunately there is a strange issue with the crystals that were used and they don't function correctly when the the device is powered on. They should oscillate at 32,768 Hz. When first turned on they often do not start oscillating at all. I have found that if you temporarily ground the terminal on the crystal normal operation will start and continue until the power is turned off. The easiest way to make sure this works is to start this program and watch the crystal oscillations. Be sure to set the delay to 1000 ms (1 second). When the crystal is not working correctly you will see something like 6,000 oscillations. Simply touch the crystal with your finger while this program is running. Short it by touching both leads with the same finger. You should see the oscillation count change to close to 33,000.
It is suggested that this be done before any rate measuring program is used so that there will be no question that the results are accurate.

11.2 C and C++

Although it is convenient to use the graphical interface of LabView the reality is that it is simply slower and less efficient than C. Some of the longer tests have been been rewritten in C so that the time required to run them could be greatly reduced. There are also many tasks that must be performed with the data after it has been collected. C++ is the language of choice for handling all of this. These tasks range from database population to data analysis.
The source code for all of these programs is located in:

/home/cfduser/CFDdata/source/

It is all organized according to function. The code itself is documented with Doxygen. It can be found in the Internal site on the level one trigger website.

11.2.1 Data Gathering Tests

The remaining tests use C programs that are run from the command line. They include: DAClinearity, DACBthreshold and RFBlinearity. They are located in the following directory:

/home/cfduser/CFDdata/bin/

This directory is also in the PATH so the commands can be run from anywhere.

DAClinearity
See the how-to.
Here is the output of the program if you do not give it any arguments:

```
[troche@cfd troche]$ DAClinearity
usage: DAClinearity [-l <lLimit> -u <uLimit> -s <step> -n <samples>] -p
<port> -i <CFDid>

 -l <lLimit>    Lower DAC limit
 -u <uLimit>    Upper DAC limit
 -s <step>      Step between DAC values
 -n <samples>   Number of Samples at each DAC value
 -p <port>      Serial port to use [MOXA channel 1-15] integer 1-15
 -i <CFDid>     ID of the CFD being tested

[troche@cfd troche]$
```

You may test up to 15 boards at a time by running multiple instances of this program. This will test both width and threshold DACs

DACBthreshold
Please see the how-to.
Here is the output of the program with no arguments:

```
[troche@cfd serial]$ DACBthreshold
usage: DACBthreshold [-l <lLimit> -u <uLimit> -s <step> -n <samples> -d
<delay> -f <frequency>] -p <port> -i <CFDid>

 -l <lLimit>    Lower DACB limit
 -u <uLimit>    Upper DACB limit
 -s <step>      Step between DACB values
 -n <samples>   Number of Samples at each DACB value
 -d <delay>     miliseconds of delay during rate calculation
 -f <frequency> Frequency of oscillator crystal
 -p <port>      Serial port to use [MOXA channel 1-15] integer 1-15
 -i <CFDid>     ID of the CFD being tested

[troche@cfd serial]$
```

You may test up to 14 boards at a time with this program. Do not use the special CTB with 75 Ohm impedance. You will probably never need to adjust the delay. Please have a good reason for doing so if you do. It will greatly increase the time the test takes if you set a high delay. Also, if you set the delay too high the results will not be good. The counter on the ADR2000 board is linked to the crystal oscillator. It can only has 16 bits of memory or 65536 counts. If you set a delay of 2 seconds or above it will loop during measurement. The delay is set to 100ms by default. If you need to adjust the frequency you should probably check the oscillator circuit. The crystals are fairly consistent once correct operation has begun. This test requires a 300mV pulse at 1 MHz be input at each board. Use the linear fan out and signal generator to accomplish this.

RFBlinearity
Please see the how-to.

Here is the output of the program with no arguments:

```
[troche@cfd serial]$ RFBlinearity
usage: RFBlinearity [-l <lLimit> -u <uLimit> -s <step> -n <samples>] -p
<port> -i <CFDid>

 -l <lLimit>    Lower RFB limit
 -u <uLimit>    Upper RFB limit
 -s <step>      Step between RFB values
 -n <samples>   Number of Samples at each RFB value
 -p <port>      Serial port to use [MOXA channel 1-15] integer 1-15
 -i <CFDid>     ID of the CFD being tested

[troche@cfd serial]$
```

This test can be performed on 14 boards at a time. Don't use the 75 Ohm impedance CTB. This test requires that you input a 1V pulse at 100 MHz to each board. Use the linear fan out and signal generator to accomplish this.

11.2.2 Data Handling

There are two aspects of data handling: Organization and Analysis. Each is important and both are done by programs written in C++.

11.2.2.1 Organization

Data organization is crucial in the project due to the vast amount of information needed to individually calibrate 2000 circuit boards. The data gathering/testing programs handle the first stage or organization by putting the data files in the proper directories and uniquely identifying each file with a CFDid. The next step is database population. This is actually the simplest part of the testing procedure. All you have to do is run the program:

insert
Insert takes care of adding all of the data to the mySQL database on gamma1. When run it looks for any new data files, processes the information inside and inserts it in to the appropriate table along with timestamps and corresponding entries in the Info table.

comment
Comment is a simple program that allows the user to input comments about a CFD into the tblCFD_Info table. Each comment is time stamped so that a sequence of events can be extrapolated from them. Such as; CFD not functioning -> R18 was cold soldered, replaced -> CFD functioning normally.

11.2.2.2 Analysis

There are a few phases of data analysis. They are somewhat interlinked but have distinct purposes. Data from the first tests must be checked to determine if the CFD is working properly. Later the data must be analyzed to determine the coefficients needed to program the CFD at runtime. The data can also be analyzed to provide useful information about programming a cfd for testing purposes. All of these functions are handled by the proceeding programs.

linear_test

After the DAClinearity test has been performed the results need to be examined. Like the name implies, this program checks to see how well the results fit a straight line. Below is an example of this test being run:

```
[troche@cfd bin]$ linear_test
Usage: linear_test <cfdid>
[troche@cfd bin]$ linear_test 511
29878
CFD 511 threshold (DACB) fits the line: y = 1356.59 + -0.976185x
It has a chi-square value of: 0.0580838
23942
CFD 511 width (DACA) fits the line: y = -569.503 + 0.449166x
It has a chi-square value of: 0.0881964
[troche@cfd bin]$
```

As long as the chi squared values are less than 1 then the fit can be considered good. Every good fit that has been observed thus far has had a chi squared value less than 0.1. The data from the test must be inserted in to the database before this program can be used as the data it taken from it.

coeff

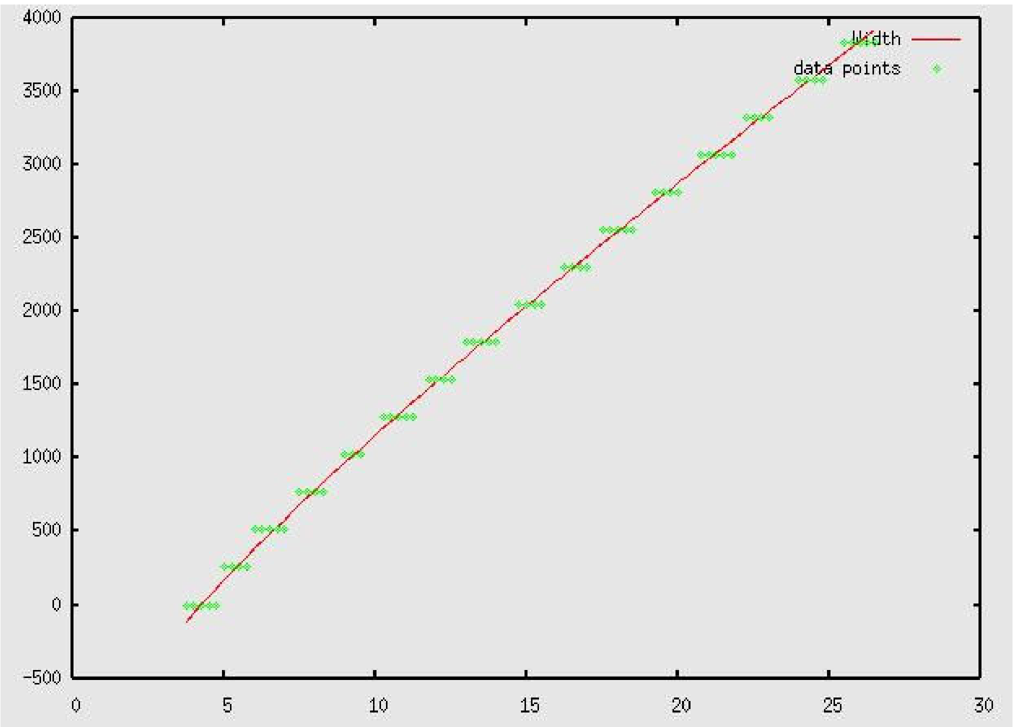
Once DAClinearity, DACAwidth.vi and RFBlinearity have been run for a particular CFD it will then be ready to have all of its coefficients calculated. In a pinch this program can be run to create default values for the coefficients if the data is not available and it will do this automatically if it doesn't find any data. Here is an example:

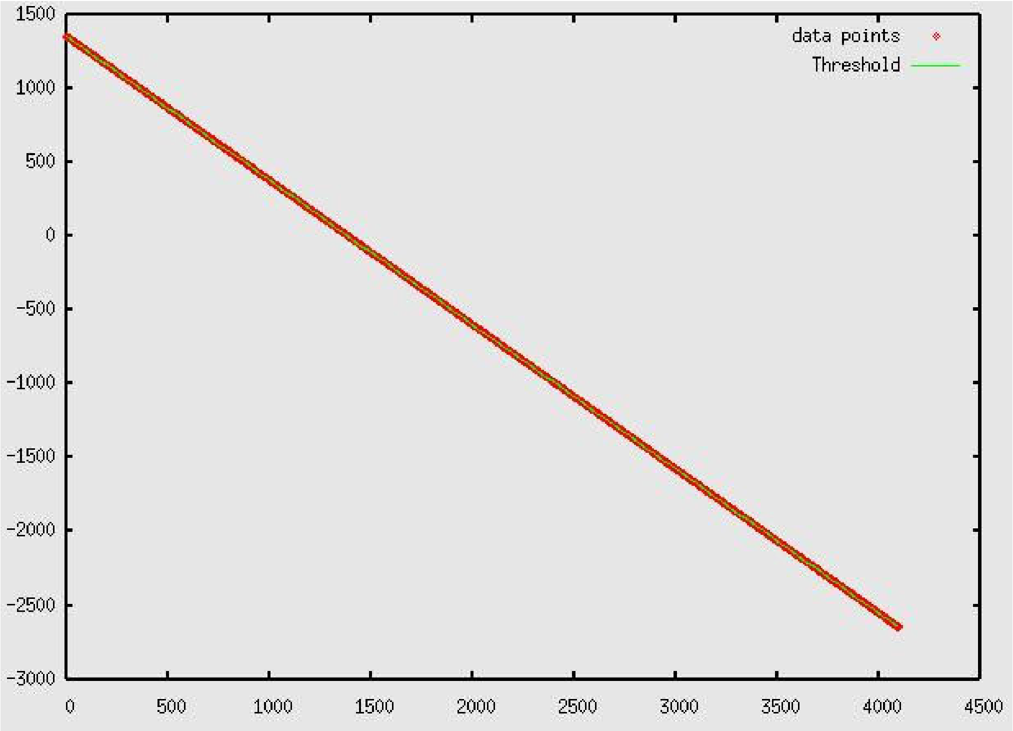
```
[troche@cfd coefficient_calculations]$ coeff

Usage: coeff <cfdid1> <cfdid2> ...

[troche@cfd coefficient_calculations]$ coeff 511 499 511 Threshold: A =
1356.59, B = -0.976185
511 Width: A = 1145.07, B = 491.708, W = w_0
511 RFB: Scale = 38.1549, Pole = 149.072

Coefficients for 511 have been calculated

499 Threshold: A = 1362.88, B = -0.976896
499 Width: A = 1897.5, B = 525.013, W = w_0
No RFB Data was found for CFD: 499
499 RFB: Scale = 36, Pole = 145

Coefficients for 499 have been calculated

[troche@cfd coefficient_calculations]$
```

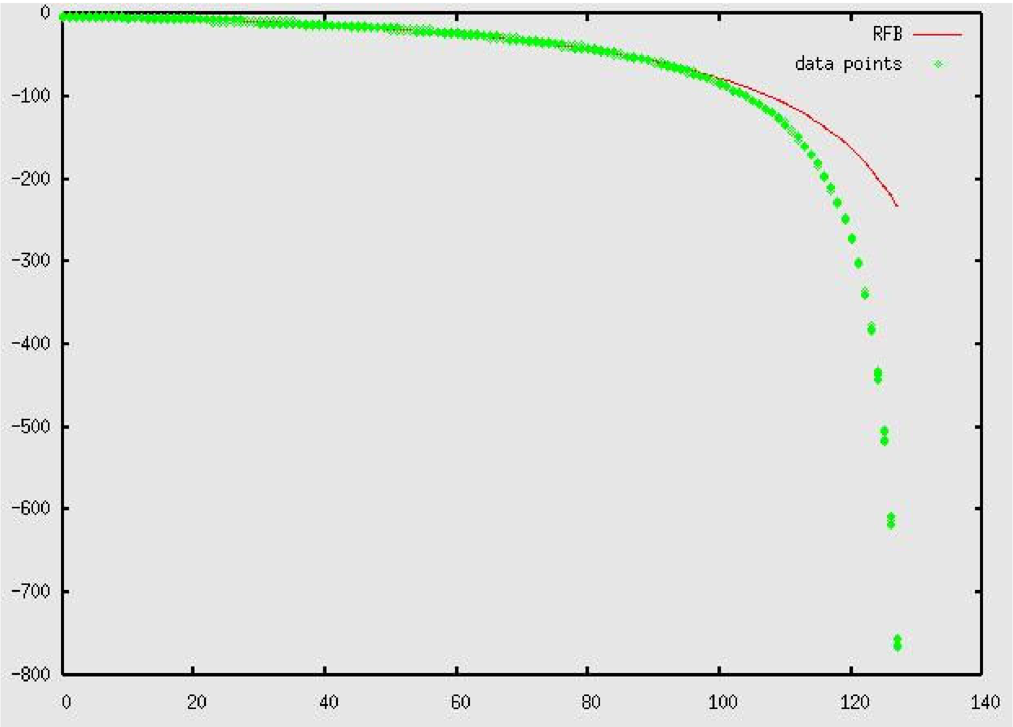
You can run coeff for as many CFDs at a time as you want.

plot_dac

This program will plot curves using the calculated coefficients for the specified CFD along with the corresponding data points. It may be used as a quick visual check for the accuracy of the fit. Below is an example of the plots it produces.




data points
Threshold
1500
1000
500
0
-500
-1000
-1500
-2000
-2500
-3000
0
500
1000
1500
2000
2500
3000
3500
4000
4500


RFB
data points
0
-100
-200
-300
-400
-500
-600
-700
-800
0
20
40
60
80
100
120
140

rfb_value

This program will allow you to quickly determine the POT setting for a desired RFB. It asks for input in mV/MHz and returns an integer value to program in to the CFD.
Example:

```
[troche@cfd init.d]$ rfb_value
Please enter the CFDid of the board in question: 511
Now please enter the rfb value in mV/MHz that you desire: 14

Coefficents for CFD 511:
Scale: 38.1549
Pole: 149.072
Chi squared: 4.58447e-06

Your POT setting for CFD number 511 at 14ns is: 40

[troche@cfd init.d]$
```

thresh_value

This program will allow you to quickly determine the DACB setting for a desired threshold. It asks for input in mV and returns a digital value just like the rfb_value. Be sure to use the proper sign for your input (most of the time you want to use a negative number). This is especially useful in determining where zero is supposed to be according to the DAClinearity test.
Example:

```
[troche@cfd CFDdata]$ thresh_value
Please enter the CFDid of the board in question: 511
Now please enter the threshold value in mV you desire: -75

Your DACB setting for CFD number 511 at -75mV is: 1465

a = 1356.59
and b = -0.976185
Chi square = 0.25924
[troche@cfd CFDdata]$
```

width_value

Like the other two value programs this one does the same thing for the width. Input in nanoseconds, output is an integer to program the CFD with.
Example:

```
[troche@cfd CFDdata]$ width_value
Please enter the CFDid of the board in question: 511
Now please enter the width value in nanoseconds that you desire: 15

Your DACA setting for CFD number 511 at 15ns is: 2203

a = 1729.58
and b = 476.782
and w_0 = 4.1
1.09426e+07
[troche@cfd CFDdata]$
```

11.3 Data Delivery

11.3.1 Summary

The coefficients calculated from the data gathered in the tests must be replicated in the database at the site. Unfortunately it is behind a firewall and there is no direct access to the database from the outside world. To accomplish this task an intricate system of shell scripts, ssh commands and file transfers coupled with C++ programs on each end have been implemented. But not to worry. Everything is accomplished by typing a simple string at the command line. As of right now there is a temporary system in place used for programming the CFDs. Instead of loading the information from the database there is a text file for each CFD with programming information. There is a program that allows the user to set the values. That will be covered in this section as well.

11.3.2 Programs Involved

This will just be the list:

SyncCoeff.pl
get_local.sql
get_remote.sql
UpdateCoeff
update_coeffs_textfile.pl
cfd_config_maker_interactive.pl

11.3.2.1 Locations

cfd.astro.ucla.edu:
- **SyncCoeff.pl**
- **get_local.sql**
- **get_remote.sql**

veritasc.sao.arizona.edu (vdaq.vts):
- **UpdateCoeff**
- **update_coeffs_textfile.pl**
- **cfd_config_maker_interactive.pl**

11.3.2.2 Functions

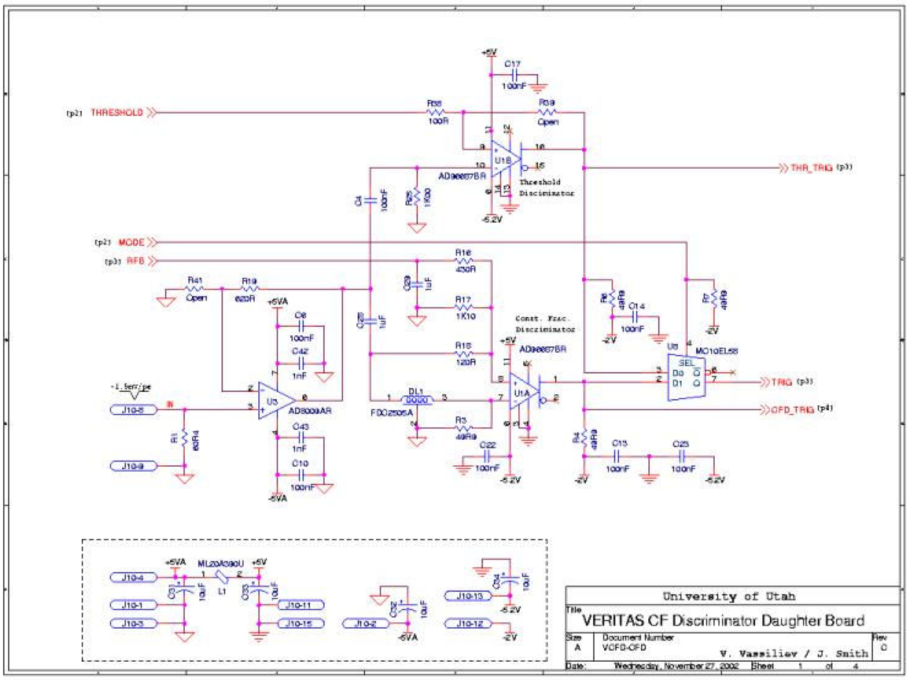
SyncCoeff.pl

This is the only program that the user needs to use when wanting to update the coefficients on the VERITAS database. It uses **get_local.sql** to query the database on gamma1.astro.ucla.edu and retrieve all of the current coefficients. It then connects to the private network at the site with ssh and queries the VERITAS database on the harvester. Shared keys are used for authentication so no passwords are necessary. Using **get_remote.sql** it gets the coefficients there. Then, by comparing creation dates it determines if any CFDs need to be created or updated. A file is created on the local machine and sent, via ssh, to the vdaq machine. The C program **UpdateCoeff** is then run which inserts the new data. Finally, **update_coeffs_textfile.pl** is run on the vdaq machine which queries the VERITAS database and updates the text file that store the coefficients for use with the temporary system.

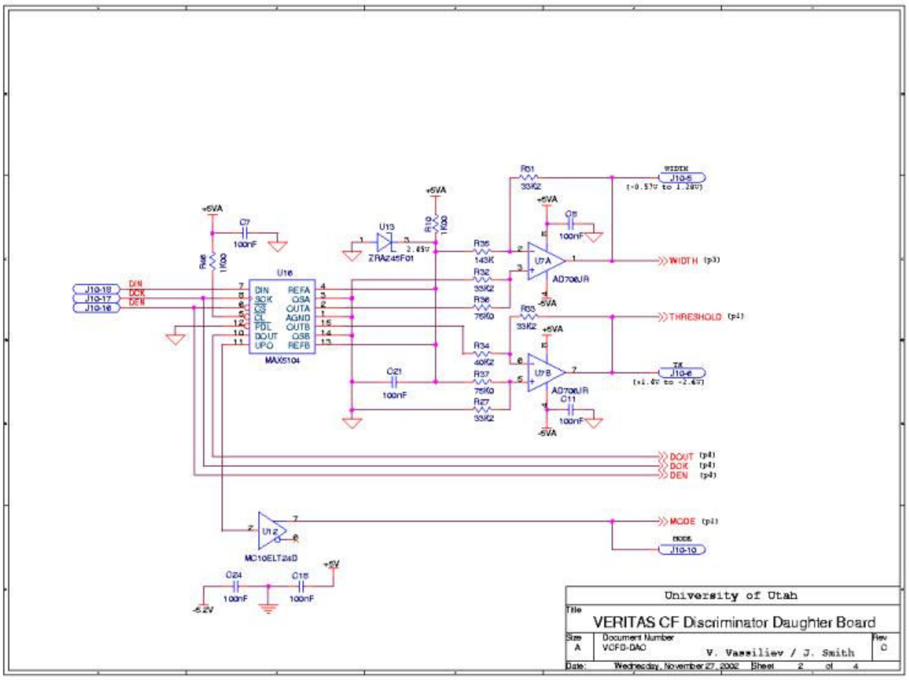
cfd_config_maker_interactive.pl

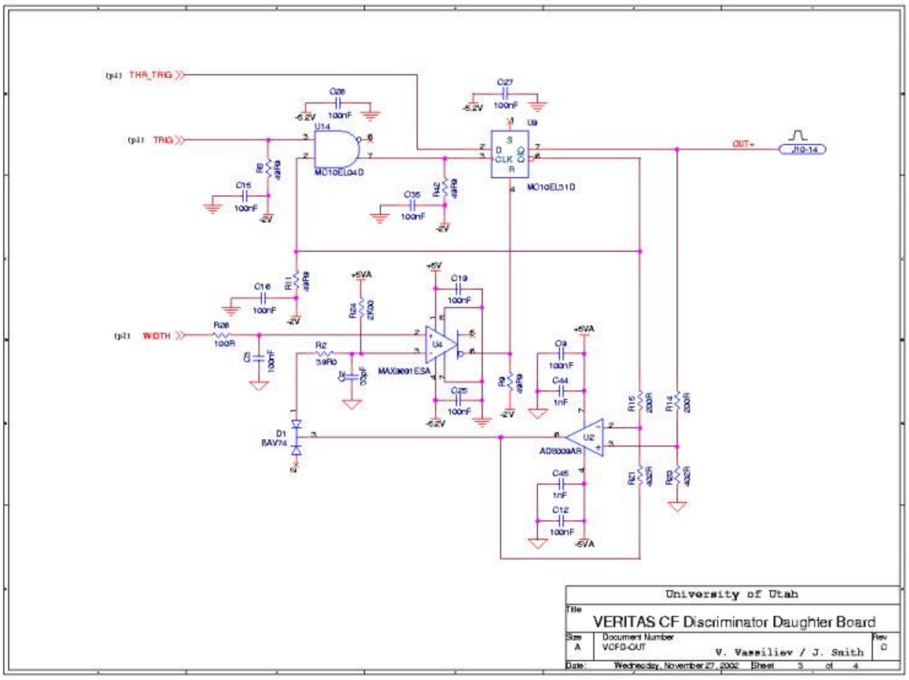
This program is used to set the values for width, threshold and rfb on the telescope. It is very self explanatory. When run it asks the user for physical values, translates them into their corresponding digital values and creates the configuration files needed to program the telescope. After an observer runs this program they should be sure to run **Propagate_Configs.pl** which transfers the files to the individual crates.

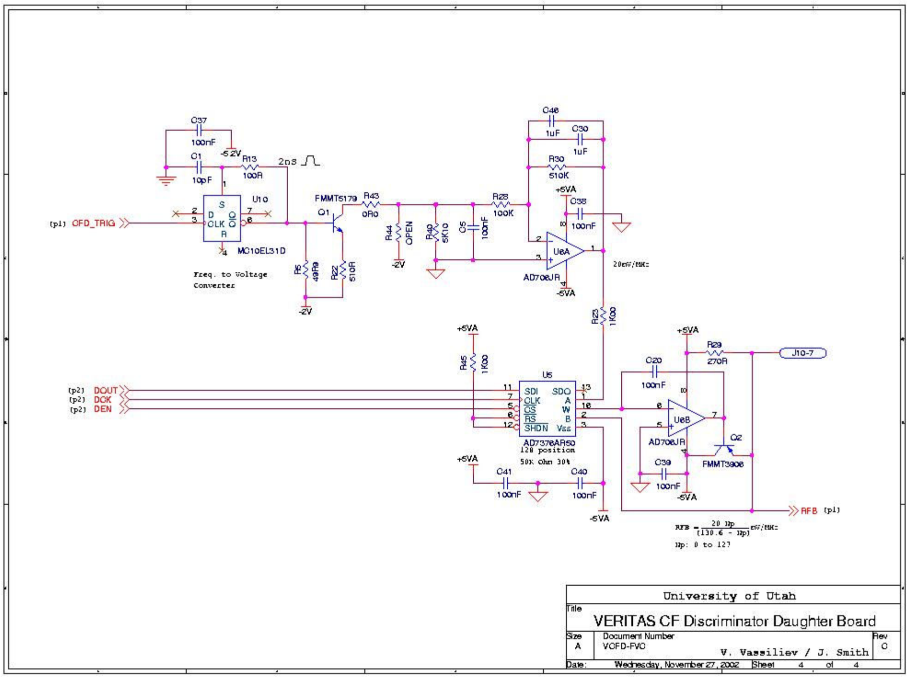
12 Schematics



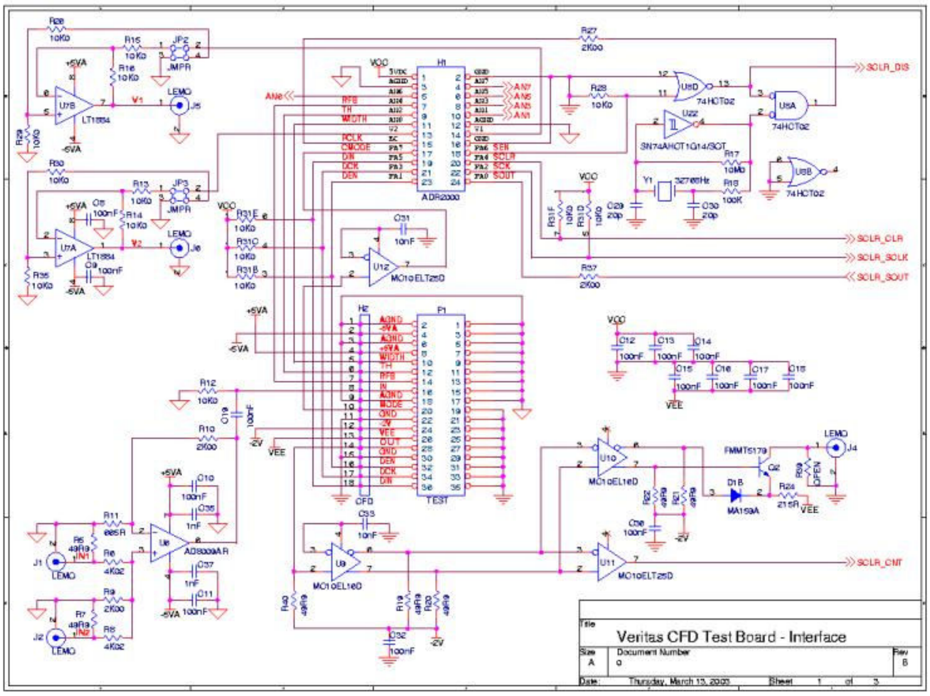
12.1 CFD

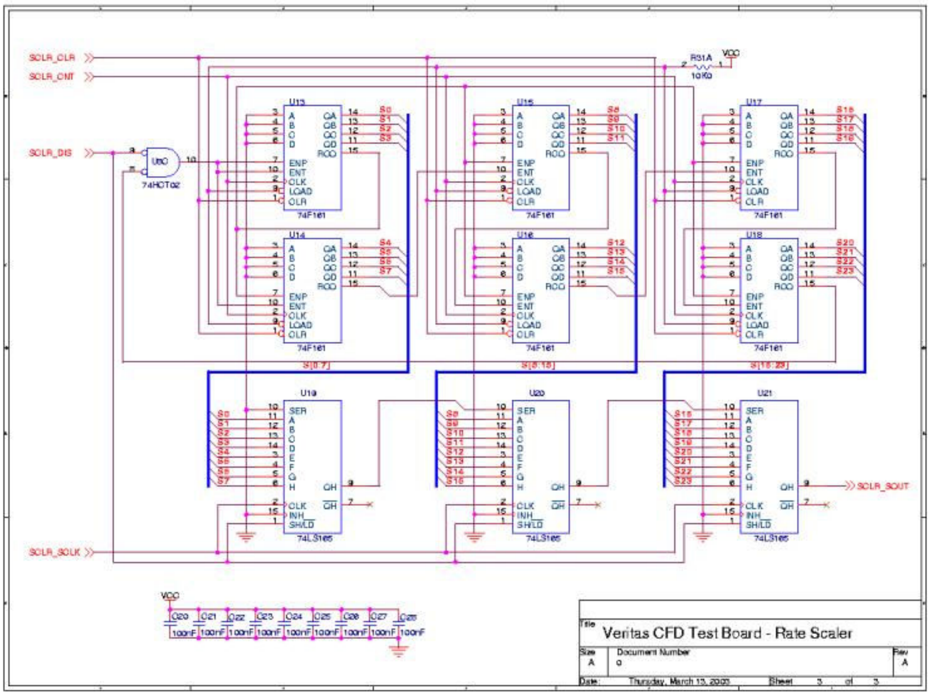

+5VA
C7
100nF
R26
1K00
U16
J10-18
J10-17
J10-16
DIN
DCK
DEN
DIN
SCK
CS
CL
PDL
DOUT
UPO
REFA
OSA
OUTA
AGND
OUTB
OSB
REFB
MAX5104
U13
ZRA245F01
2.45V
+5VA
R10
1K00
R35
143K
R32
33K2
R36
75K0
R34
40K2
R37
75K0
R27
33K2
C21
100nF
R31
33K2
WIDTH
J10-5
(-0.57V to 1.28V)
+5VA
C5
100nF
U7A
AD706JR
-5VA
WIDTH (p3)
R33
33K2
THRESHOLD (p4)
+5VA
U7B
AD706JR
C11
100nF
-5VA
TX
J10-6
(+1.0V to -2.6V)
DOUT (p4)
DCK (p4)
DEN (p4)
U12
MC10ELT24D
MODE (p4)
MODE
J10-10
C24
100nF
-5.2V
C18
100nF
+5V
University of Utah
Title
VERITAS CF Discriminator Daughter Board
Size
A
Document Number
VCFD-DAC
V. Vassiliev / J. Smith
Rev
C
Date: Wednesday, November 27, 2002 Sheet 2 of 4

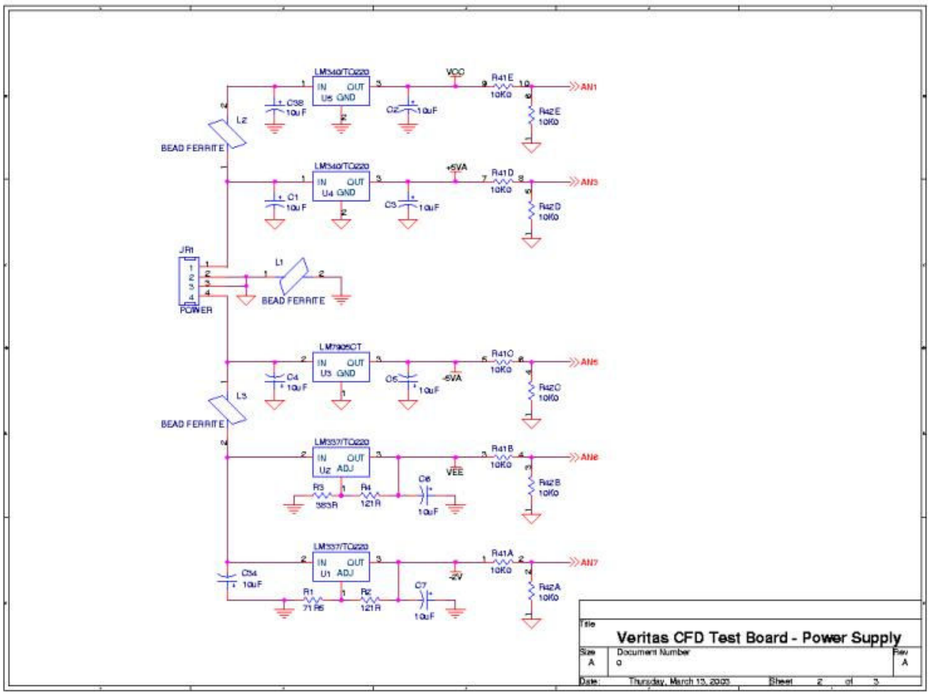

(p4) THR_TRIG
(p4) TRIG
C28
-5.2V
100nF
U14
MC10EL04D
R8
49R9
C15
100nF
-2V
C27
-5.2V
100nF
U9
S
D
CLK
R
Q
MC10EL31D
OUT+
J10-14
C35
100nF
R42
49R9
-2V
C16
100nF
R11
49R9
-2V
+5VA
R24
2K00
+5V
C19
100nF
(p2) WIDTH
R28
100R
C3
100nF
R2
39R0
C2
33pF
U4
MAX9691ESA
C25
100nF
-5.2V
R9
49R9
-2V
+5VA
C9
100nF
C44
1nF
D1
BAV74
U2
AD8009AR
R15
200R
R14
200R
R21
432R
R23
432R
C45
1nF
C12
100nF
-5VA
University of Utah
Title
VERITAS CF Discriminator Daughter Board
Size
A
Document Number
VCFD-OUT
V. Vassiliev / J. Smith
Rev
C
Date: Wednesday, November 27, 2002 Sheet 3 of 4



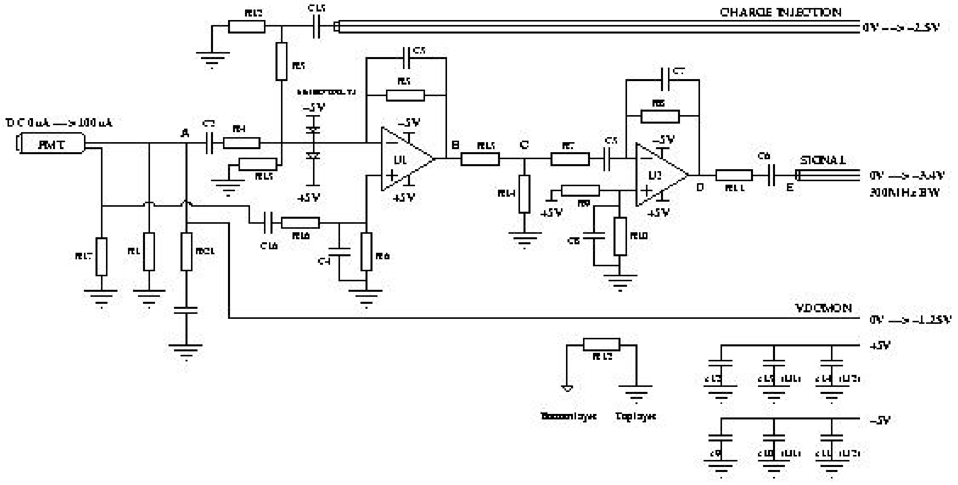
12.2 Test Board







12.3 PMT Amplifier


CHARGE INJECTION
0V --> -2.5V
R12
C15
R3
C3
R5
R4
-5V
DC 0uA ---> 100uA
PMT
A
C2
R13
+5V
U1
-5V
+5V
B
R15
C
R7
C5
C7
R8
-5V
U2
+5V
D
R11
C6
E
SIGNAL
0V ---> -3.4V
300MHz BW
R14
+5V
R9
C8
R10
R17
R1
R2
C16
R16
C4
R6
VDCMON
0V ---> -1.25V
R12
Bottom layer
Top layer
+5V
c12
c13 (U1)
c14 (U2)
-5V
c9
c10 (U1)
c11 (U2)