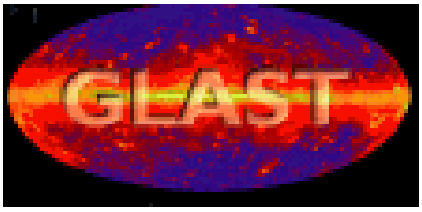
GLAST LAT ACD SUBSYSTEM DOCUMENT	Document # ACD-RPT-000390 Rev	Date Effective 9-16-2005
	Prepared by: Karen Calvert	Supersedes
Document Title: GLAST LAT ACD EGSE Software and Test Information		

Gamma-ray Large Area Space Telescope (GLAST)
Large Area Telescope (LAT)
Anti-Coincidence Detector (ACD)
EGSE Software and Test Information

ACD-RPT-000390
Rev



Goddard Space Flight Center
Greenbelt, MD 20771

1. Introduction

This document was compiled primarily by Karen Calvert as a working description of the ACD test environment at Goddard. Although there are places where the text remains incomplete, it contains much useful information about the operation, constraints, and troubleshooting of the ACD.

2. Description of ACD EGSE and Test Environment

ACD ENVIRONMENT

Environment Variables

ACD_ROOT: As can be seen from our acd.cfg file, we set the ACD_ROOT directory to the location of our 'ACD' directory.
ACD_PATH: We use this to direct our output - during our testing we have used the following layout:
ACD_PATH = c:\ACD_OUTPUT (for ACD level testing)
ACD_PATH = c:\ACD_OUTPUT\Acd-Karen (for s/w devel & testing).
Under this level, we use the following self-explanatory sub-directories:
$ACD_PATH\export
$ACD_PATH\snapshots
$ACD_PATH\reports
$ACD_PATH\data
$ACD_PATH\logs
$ACD_PATH\fits
ACD_WORK: This is used by the offline schema generation software and is not needed for testing the ACD under the runControl environment.

NOTE: To date, we have never used FITS as it was not sufficiently formatted to for our tools to ingest in any meaningful way. Instead, the scientists had a tool developed to ingest and analyze .ldf files.

Batch Files

For ACD level testing, we have two batch files defined:

1) acd-gitot8_pri.bat	Loads the primary side schema file
2) acd-gitot8_red.bat	Loads the redundant side schema file

ACD SCRIPTS & SOFTWARE DIRECTORY STRUCTURE

$ACD_ROOT	ACD python code used to support ACD testscripts
$ACD_ROOT/qt	ACD Qt python code used for GUI-based applications as well as simple dialog prompts.
$ACD_ROOT/testscripts	Any files at this level are likely to be obsolete or at least extremely dusty.
$ACD_ROOT/testscripts/devel	This directory was used as a temporary staging area for testscripts that were not fully tested or were in the development/modification phase. This was done to provide an easy way to provide everyone with access without compromising a working set of scripts.
$ACD_ROOT/testscripts/release	The files in this directory represent the deliverable testscripts and suites. All scripts can be run standalone or are callable from a suite. The following suites are valid and were used during ACD Environmental Testing - from these suites, all valid subscripts can be referenced: • AcdAlivenessTest.py - Basic aliveness test that checks power/current, event readout w/TCI charge, and performance of PMT's using the HVBS. • AcdClockMargin.py - Tests basic performance and timing

when operating at specific clock frequencies
- AcdGafeTests.py - Tests performance of GAFEs
- AcdInitPowerUpScripts.py - Verifies correct 3.3V and 28V power consumption (also with primary and secondary drivers)
- AcdLongFunctional.py - Detailed ACD functional test suite. Takes approx. 2.5 hours to run
- AcdRedundantTest.py - Abbreviated form of AcdAlivenessTest to test functionality using secondary power
- AcdVoltMargin.py - Tests basic performance when operating at specific voltage levels.

The following data collection scripts are run routinely to verify ACD performance.
- AcdTriggeredOp.py - ACD triggered data collection script using ACD triggers from ROI coincidences. Script runs until operator presses STOP. HVBS set to normal mode levels.
- AcdOpExtTrg.py - External triggered data collection script that uses a pair of coincident paddles to trigger on vertical (more or less) muons. Runs until operator presses STOP. HVBS set to normal mode levels.
- AcdMonitor.py - ACD Performance monitoring script that monitors h/w rates from single-channel triggers and optionally records event data (when not being used for light-tight testing). Sets HVBS to normal mode levels Script runs until operator presses STOP.

$ACD_ROOT/Schemas Schema and configuration files. The valid schema files are:
- AcdSchemaFullAcdPrimary.xml - For primary side testing
- AcdSchemaFullAcdRedundant.xml - For redundant side testing

The following configuration files are included from the above schemas:
- envMonConfigFullACDPrimary.xml ACD related environmental monitor configuration for primary side testing
- envMonConfigFullACDRedundant.xml ACD related environmental monitor configuration for redundant side testing

All but the last two configuration files listed below are (or should be) included from the two schema files listed above. These files aren't too pertinent because all calibrations are loaded from a table residing in the AcdFreeCal.py file. Also, they aren't optimally designed as they contain redundant EGU and constraint definitions among other possible deficiencies.
- AcdConfigChassis1L
- AcdConfigChassis1R
- AcdConfigChassis2L
- AcdConfigChassis2R
- AcdConfigChassis3L
- AcdConfigChassis3R
- AcdConfigChassis4L
- AcdConfigChassis4R
- AcdConfigChassis5S (spare flight chassis)
- AcdConfigQual (qualification chassis)

$ACD_ROOT/Schemas/builder Software to automatically generate Schema files.

$ACD_ROOT/Schemas/builder/configuration Contains Long Functional baseline data that is used to compare against runs. To date, a firm baseline has not been established

$ACD_ROOT/Schemas/builder/configuration/spreadsheets Contains the calibration data from

LabView-based FREE card and chassis bench testing

ACD CONFIGURATION - Schema Files and AcdFreeCal

AcdFreeCal

Due to the dynamic nature of Chassis testing with regards to GASUs being used and availability of ports on those GASUs (i.e. mapping of GARC to port issue) we decided to implement a lookup table for setting ACD registers to calibrated values. The table is defined in the $ACD_ROOT/AcdFreeCal.py file. The register settings are referenced by GARC serial number, temperature, and voltage. The majority of the table was populated by hand in a cut-and-paste fashion from spreadsheet data generated from LabView calibration testing performed on individual FREE cards and chassis' before the G3 teststand was delivered to GSFC. Other timing related data was derived through testing under various environments using the G3 teststand during Chassis level testing. Every script that requires calibrated settings sets the registers using the table in the AcdFreeCal.py file.

Schema Files

Final calibrated settings will be available _after_ all offline data analysis has been completed. These will be provided via properly constructed Schema and configuration files. NOTE: The current schema configuration files residing in the Schemas directory for each Chassis are not very well constructed nor relied on for proper calibration of the ACD instrument. In the near future, the ACD team will provide valid Schema files that document proper configuration of the ACD under various operating environments (i.e. temperatures, voltages, and frequencies). The software to produce these Schema files resides in the $ACD_ROOT/Schemas/builder as documented above. The input to the builder software comes from four sources:

1. LabView Calibration in the Form of .csv files
2. Manually Derived .csv Files Resulting from Data Analysis
3. Script Generated .csv Files
4. Fixed Default Values

LabView Calibration - This includes settings for the following registers:

- -
- -
-

Derived from Data Analysis - This includes settings for the following registers:

- HV_NORMAL - voltage levels for normal (i.e. non-SAA) mode. (Alex)
- BIAS_DAC - need to derive optimal settings over all temperature ranges. (Bob H.)
- VETO_DAC - comes from analysis of pha data from threshold scan using AcdTriggeredOps test. (Alex/Bob). EGUs now are for pC, but will be modified to go from raw to pC to MIPs.
- VERNIER_DAC- comes from threshold scan analysis of pha data from threshold scan using AcdTriggeredOps test. (Alex/Bob). EGUs now are for pC, but will be modified to go from raw to pC to MIPs.
- HLD_DAC - from data analysis - Alex/Bob. EGUs now are for pC, but will be modified to go from raw to pC to MIPs.

Script Generated - This includes settings for the following registers/derived parameters:

- HITMAP_DELAY Calculated by AcdHitmapDelay.py (solicited) and AcdHitmapDelayACD.py (ACD triggered)
- HOLD_DELAY Calculated by AcdHoldDelay.py (solicited) and AcdHoldDelayACD.py (ACD triggered)
- PHA_THRESHOLD_0 - 17 Calculated by adding 15 to AcdPedestal mean PHA calculations. NOTE: Need to take temp variation into account (i.e. might be different offset at low & high temps)
- ENV_HV1_<port> EGU Calculated by AcdHvbsPmAlive.py for all ports.
- ENV_HV2_<port> EGU Calculated by AcdHvbsPmtAlive.py for all ports
- Need to add derived/output parameters that are stored as opaque data
  -

Fixed Default Values - These are stored in the builder software garcDefault.cfg file. This includes settings for the following registers:

- VETO_DELAY = 0
- VETO_WIDTH = 2
- HITMAP_DEADTIME = 3
- ADC_TACQ = 0
- MAX_PHA = 6
- MODE = 0x300
- SAA = 300V
- VETO_EN_0 = 0xFFFF
- VETO_EN_1 = 0x3
- PHA_EN_0 = 0xFFFF
- PHA_EN_1 = 0x3

Operational Schema vs ACD Test Schema

PHA Enable: Since ACD tests involve checking all channels whether populated or not, our default setting for the PHA Enable registers (PHA_EN_0 and PHA_EN_1) is to have all PHAs enabled. Most all scripts should explicitly set this anyway although some may not. Operationally, unpopulated channels should be disabled, therefore, the Schema configuration files should have GARC unique settings for these two registers. PHA_EN_0 sets channels 0 (bit 0) – 15 (bit 1) and PHA_EN_1 sets channels 16 (bit 0) and 17 (bit 1).

VETO Enable: Same as above for PHA Enables.

Generating Schema Files with the Schema Builder Tool

To run the builder software to generate all required Schema files, execute the batch file AcdSchemaGen.bat. Set the $ACD_WORK environment variable within the batch file to point to the root location of the baseline .csv files and spreadsheet data directory. The setup of these directories should be as follows:

$ACD_WORK/Configurations - Contains the baseline .csv files and the garcDefaults.cfg file. The .csv files are the script output parameter files that have been established as representing the baseline configuration.

$ACD_WORK/Configurations/Spreadsheets - Contains all of the LabView calibrated data converted from Excel spreadsheets to .csv formatted files.

In the ACD GSE setup, we committed the baseline configuration and spreadsheet files to the CVS repository under the $ACD_ROOT/Schemas/builder directory. We don't have installation software, so we just perform a CVS checkout and use that as our $ACD_ROOT.

SCRIPT OUTPUT

Test Reports

Each script, whether run by itself or from a suite, produces a test report. A test suite produces a series of nested test reports where each report has links to lower level reports generated from sub-suites or sub-scripts within the suite. Each test report is named as follows: <script-name>_<seq>.html, where <seq> is sequence number used to prevent overwriting of previous runs. For example, the AcdPedestal.py test may produce a test report named AcdPedestal_5.html.

All test reports include a 'PASSED'/'FAILED' completion status. When run from a suite, any test that fails requires a Sign-Off. The sign-off area is provided in the suite level test report.

Snapshots

Scripts that are run within a suite produce a pair of snapshot files that are equivalent to the runControl rsa and rsb snapshots. The snapshots are named as follows <script-name>A.html and <script-name>B.html. For example, when run within a suite, the AcdPedestal.py test produces the snapshot files AcdPedestalA.xml and AcdPedestalB.xml

Output Parameters (.csv files)

Most scripts (at least those that are included in the Long Functional suite) produce ASCII, comma delimited files containing output parameters calculated by the script. These output parameters can later be analyzed using the AcdBaselineCompare.py script. More on this below.

Command Prompt Messages

All scripts output informational type messages to the command prompt window. These used to stem from echoes from the logger, but we felt that the voluminous nature of these messages was impacting memory utilization and causing problems. In many cases, the log.info calls were changed to print statements instead. Because some of our scripts can take quite long to execute (due mostly to lack of hardware scalars), these messages help to provide a warm-fuzzy that the script is actively running and doing what it is supposed to do.

TEST RESULTS

Routine Failures

Most tests consistently result in a 'PASSED' completion status, however, the following tests routinely fail, but do not pose a concern:

1. **AcdGafeNoise.py** - The failures involve the inability to calculate the vernier sharpness on some channels. This is due to a less than robust algorithm for determining threshold points. These failed channels are not consistent between runs. A visual inspection of the results shown in the test report has always been performed to validate that a better algorithm would find these threshold points and the sharpness could then be calculated. Due to other more pressing matters, the scientists chose to keep this a manual operation rather than specifying a better algorithm. Vernier sharpness is calculated by taking the difference between VERNIER settings that produce between 10% +/- 5% and 90% +/- 5% of the TCI rate. This is a measure of how GAFE or system noise smears the VETO threshold. Sometimes, the actual rate exceeds this criteria by just a couple of percent.
2. **AcdHldCal.py** - This test consistently fails to determine the 50% efficiency HLD_DAC settings for all channels of GARC 5 when the test is run on the entire ACD system. If run on just GARC 5 or some subset of the GARCs, the test is successful. This is an unexplained phenomenon. We routinely rerun this test by itself following the Long Functional test at which point it always passes.
3. **AcdTriggeredOps.py** - In a 2-3 hour run, this test will experience 8-12 ASC solicited event timeouts. Since this is a data collection script, this does not impact its performance.
4. **General Event Timeout Issues** - Apparently we still occasionally, albeit rarely, experience event timeouts that cause script failures. This is newly acquired knowledge (for me) and it needs to be investigated further.

Occassional Failures

These failures occur due to readbacks being out of expected range. In these cases the readbacks vary over temperature and the ranges may need to be expanded. Currently, expected ranges are stored in the file AcdConstants.py and there is no provision for environmental changes (i.e. different ranges for different temperatures). This problem may go away when temperature specific schema files that will implement range checking through rules are generated and loaded accordingly. However, some failures occur on derived parameters rather than physical registers and we don't know what provisions are in place to perform limit checking on these (other than ferencing a named constraints

ourselves - can opaque data have links to rules/constraints/EGUs/etc?

OTHER TESTS/UTILITIES

The following tests/utilities have recently been developed to provide additional data analysis and troubleshooting capability.

Baseline Comparison - Compares testscript output written to .csv files against the established baseline (also stored in .csv files). Actually, any set of files may be used in the comparison (eg. test results from two different runs). The user is prompted for the directory location of the two sets of files to be compared. The script compares all .csv files it finds in the given locations. Most every long functional script generates a temperature/voltage unique .csv file for each GARC it is run against.

Snapshot Comparison - Compares the before and after snapshots from a specified run. Only the GARC and GAFE registers are examined. A test report is generated showing all before and after values with any deviations highlighted in bold type. If there are any deviations in values of the CMD_REJECT register or the MODE register an error is recorded. Unfortunately, many registers routinely (and purposely) change during the execution of a script. These changes are dependent on the test the script is performing. This makes it a bit tricky to quickly detect anomalies and we did not spend the time to add the smarts to take into account which script produced the snapshot. However, since this script is used for trouble-shooting purposes, it still provides a great quick look capability to spot obvious problems. The following obvious anomalies can be spotted:

- CMD_REJECT Register This GARC register is used to count the number of rejected commands. Ideally, this should never increment. It is an 8-bit counter so it rolls over after receiving 255 bad commands. This counter is reset only after a GARC reset or power recylcle.
- TCI_DAC Register: For certain scripts, this register should have a known value and should never be set to zero. We detected this condition recently while troubleshooting a AcdFullGasuHwCtrs.py test failure.
- MODE register This register should always be set to it's default value of 0x300.

GARC RETURN DATA PHASE TESTING

The ACD has the ability to adjust the return data phase that it uses. By default, the ACD uses the positive edge phase, however, it runs without error using either edge therefore we do not have a test for an out-of-phase condition. The phase is controlled through bit 11 of the GARC MODE register. The default setting for the MODE register is 0x300. To set the phase to the negative edge, the mode register can be set to 0xB00 or can be ORed with 0x800. For troubleshooting purposes, the ACD H/W IF TOOL (launched from the ACD runControl tab).can be used to set the GARC MODE register interactively.

CALIBRATION

GASU UNIQUE CALIBRATION

There are a couple of EGUs related to environmental monitors that are dependent on the GASU used. These EGUs are defined in the envMonConfigFullACDPrimary.xml andenvMonConfigFullACDRedundant.xml files and include the following:

1. EGUs for the AEQ HVBS monitor readouts: env_hv1_<port> and env_hv2_<port>, where <port> represents one of 12 GASU/GARC ports (1LA - 4RB)
2. EGUs for the AEQ DAQ Digital 3.3V current monitor readouts:

To derive the HVBS EGUs, run the AcdHvbsPmtAliveness.py script and see the test report Slope and Intercept table. These values must be entered into the appropriate places in the above mentioned environmental monitor configuration files.

To derive the EGUs for the 3.3V Current monitor readouts, a voltmeter must be used to take actual measurements.

NOTE: Timing seems to be fairly consistent across all GASUs.

TIMING

Once the ACD gets integrated with the LAT, we anticipate that the trigger timing may need to be adjusted. We have several timing adjustments that can be made on the ACD. The following GARC registers can be adjusted to optimize timing for readouts: hitmap_delay, hitmap_width, hitmap_deadtime, veto_delay, veto_width . Once the proper readout timing has been determined, the GARC hold_delay register can be adjusted to optimize PHA data associated with the readout. We have the following timing related scripts that are used to calibrate hitmap_delay, hitmap_width, and hold_delay for both solicited and ACD triggering. The remaining timing related parameters are set to a default value.

- AcdHitmapTiming.py - Used in ACD test suites to verify that current timing parameters are accurate.
- AcdHitmapDelay.py - Calculates optimum hitmap delay setting for solicited triggering for each GARC
- AcdHoldDelay.py - Using a properly calibrated hitmap delay setting for solicited triggering, calculates the optimum hold delay setting for each GARC (i.e. the setting that produces the highest mean PHA).
- AcdHitmapDelay_ACD.py - Calculates optimum hitmap delay setting for ACD triggering for each GARC The output of this script is used to populate a timing configuration file to be used for ACD triggering
- AcdHoldDelay_ACD.py - Using a properly calibrated hitmap delay setting for ACD triggering, calculates the optimum hold delay setting for each GARC. The output of this script is used to generate a timing configuration file to be used for ACD triggering that should be used by any scripts using ACD triggering.

Hold Delay

Analysis has shown that a 5-step error in the hold delay produces only a tiny difference in the PHA value, no more than 1% if one of the settings is near optimum. Even if the hold delay is optimized very poorly, the error is probably no more than 5% - the worst possible case would be about a 10% error. (5 steps is roughly the span of optimum settings with in a FREE card.)

Timing Configuration Files

Two timing configuration files are created by the Schema Builder software using the output generated by the hitmap delay and hold delay timing scripts described above. One file is for ACD triggering and is included by the Schema files to establish default timing settings. The other file is for solicited triggering and should be loaded by testscripts that use solicited trigger readouts (not currently done).

AEM Trigger Sequence TACK Delay

1. ACD The optimal TACK delay setting for ACD triggering is 0.
2. SOLICITED The optimal TACK delay for solicited triggering is 80.

EXTERNAL TRIGGERING

The optimal hitmap delay for external triggering is 0.

BASIC TROUBLESHOOTING

Register Diagnostics

Here are some GARC registers you can examine for error conditions. These registers can be examined through the register browser OR the before and after snapshots (either rsaXXX.xml/rsbXXX.xml if run standalone or <script>A.xml/<script>B.xml if run from an ACD suite.)

GARC Diagnostic Register

Bit 15: parity_error
Bit 14: cmd_parity_error
Bit 13: data_parity_error
Bit 12: cmd_error
Bits 11-8: diag state loop counter
Bits 7-0: valid command counter

GARC CMD_REJECT Register

Shows number of rejected commands since last reset.

GARC LAST_CMD Register

Shows the last command received in error. This is reset only after a GARC reset or a subsequent bad command is received.

Solicited Event Timeouts

Timeout on ALL Solicited Events: After moving our teststand to the vibration facility, we encountered the inability to receive solicited events. We saw that the LCB was loose. Interestingly enough, during this condition, regular ACD register reads/writes worked fine. When the LCB was tightened, the problem disappeared.

Occassional Timeouts: We have seen two scenarios where occassional timeouts are received:

1. Our event parser randomly takes an unexplainably long time to parse an event. Normally an event might take only .01 seconds to parse, but occassionally we would see it take as much as 3 or 4 seconds. Never could figure out why, but we just increased our timeout period to 5 seconds. This is done in our waitForEvent() calls. This seemed increased timeout period seemed to "fix" the problem.
2. Our AcdTriggeredOp.py data collection script occassionally sees timeouts when waiting for ASC solicited events. It looks like the event is not just taking longer than expected like in the scenario described above, but rather that the event is not coming in at all. Perhaps a snapshot analysis may show a parity error at the LCB, AEM, or GARC level that might explain this.

NOTE: Recent analysis of some test data shows that a script (AcdGafeNoise.py) received an ASC timeout and bailed out. Not sure which scenrio above caused this problem. I believe this only happened once since we changed the timeout to 5 seconds, but I'm not sure.

Packet Error Events

If back-pressure gets exerted on the system (i.e. the event rate exceeds what the system can handle), packet errors will be generated. For our version of the teststand, this resulted in packet truncated errors. For data collection programs like AcdTriggeredOp.py, the event rate may get too high if the VETO/Vernier threshold is set too low for the HVBS level used. To resolve this problem, either lower the voltage or raise the threshold settings.

UNEXPLAINED ISSUES

We have experienced the following problems and were unable to determine the cause:

1. Event timeouts: This has already been discussed above, but a recent discovery by SLAC may have attributed this problem to a bad network driver. The connection with the bad network driver and the timeouts we experienced has not been verified. We also rarely if ever experienced this problem in our lab with the Qual Chassis (that we know of anyway).
2. Missing ACD data: The AcdTciRegRange.py and AcdTciHighRange.py scripts have experienced rare occassions where ACD data is not included in some of the events that these scripts generate. Inspection of the .ldf file concur that the data is missing. This issue has occurred with the Qual Chassis running with GASU 6 both at GSFC and SLAC.

OPERATIONAL SAFETY

HVBS Use

The high voltage bias supply (HVBS) should never be used when an ionizer is in close proximity and in active use. Also, to save on wear and tear, the GARC should not be powered down when the HVBS is enabled and the change in voltage settings should not exceed increments of 500 volts

HVBS Constraint

Due to observed tube performance at high voltage levels, the HVBS should never be set to levels exceeding 1000 volts (3150 raw).