SANDIA REPORT

SAND2016-11835
Unlimited Release
Printed November 2016

AE Recorder Characteristics and Development

Michael E Partridge, Shane K Curtis, David P McGrogan

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2016-11835
Unlimited Release
Printed November 2016

AE Recorder Characteristics and Development

Michael E Partridge, Shane K Curtis, David P McGrogan
Advanced Fuzing Technologies Department
Threat Analysis Technologies Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0661

Abstract

The Anomalous Environment Recorder (AE Recorder) provides a robust data recording capability for multiple high-shock applications including earth penetrators. The AE Recorder, packaged as a 2.4" diameter cylinder 3" tall, acquires 12 accelerometer, 2 auxiliary, and 6 discrete signal channels at 250k samples / second. Recording depth is 213 seconds plus 75ms of pre-trigger data. The mechanical, electrical, and firmware are described as well as support electronics designed for the first use of the recorder.

Acknowledgments

The quick response developing a complete high-shock on-board data recorder system in only 13 months requires a strong team effort including excellent customer support. The recorder design was based on the partially-completed 3DDR-AM recorder developed under sponsorship of Danny Hayles, retired from DTRA. The recorder housing mechanical design began with an initial approach by Matthew Neidigk on 3DDR-AM, and then refined by Shane Curtis for the AE Recorder. The many other mechanical parts were Shane's alone. The signal conditioning was developed by Cory Larsen for 3DDR-AM. David McGrogan designed the SmartFusion gate array and associated firmware, and wrote the Digital section of this report. Jay Smith wrote the LabWindows testing software for the analog board and assembled system. Jeremy Giron developed firmware for the analog and power /interface boards, and with Stephen Simpson's help, refined the firmware and Coulomb-counter hardware on the Battery Management System. David Cutler created the fiber-optic aspects of the design, and developed LabWindows software for the track-side control plus firmware for the fiber optic interface on-board the test article. Berto Jimenez assembled the flight hardware with his characteristic high precision, resulting in the demonstrated reliability of the system. The clear board and assembly photos in this report are also Berto's. Finally, the team is greatly appreciative of Matt Brewer, our customer for the system, who pushed us hard enough to produce technical excellence.

CONTENTS

FIGURES

TABLES

NOMENCLATURE

3AMP	3-Axis MilliPen penetrator data recorder, 4 channel, 75ksps, 218-msec record
3AMP Interim	Replaced 3AMP sunset components, 6 channel, 75ksps, 581-msec record
3DDR-AM	3-Axis DTRA Data Recorder – Advanced Miniaturization
ADC	Analog-to-Digital Converter
CMRR	Common-Mode Rejection Ratio
CTE	Coefficients of Thermal Expansion
DAC	Digital-to-Analog Converter
DOE	Department of Energy
DTRA	Defense Threat Reduction Agency
GMB	glass micro-balloon, a fill material for epoxy encapsulation
HiCapPen	High-Capacity Penetrator data recorder, 19 channel, 150ksps, 100-sec record
ksps	kilo-samples per second
MFB	Multiple Feedback
PGA	Programmable Gain Amplifier
SMPC	Switch-Mode Power Converter
SNL	Sandia National Laboratories
SPI	Serial-Programmable Interface

INTRODUCTION

The Anomalous Environment Recorder (AE Recorder) provides a robust data recording capability for high-shock applications such as earth penetrator instrumentation and impact testing. The expanded-capability, high-fidelity AE Recorder acquires 12 accelerometer, 2 auxiliary, and 6 discrete signal channels at 250k samples / second with more than 213 seconds recording depth, plus 75ms of pre-trigger data. The small, 2.4" diameter design is also compact enough to support sub-scale penetrator tests including fuze development.

A challenge designing on-board instrumentation for the B61-12 anomalous environment test series was addressing operational safety. Following the 2008 sled track accident [1] (1) (2), sled track tests at Sandia involving on-board instrumentation were essentially eliminated because energy necessary for operating instrumentation could inadvertently fire energetic devices. Restarting instrumented sled test operations required reconsideration of all test processes, with additional analysis and mitigation.

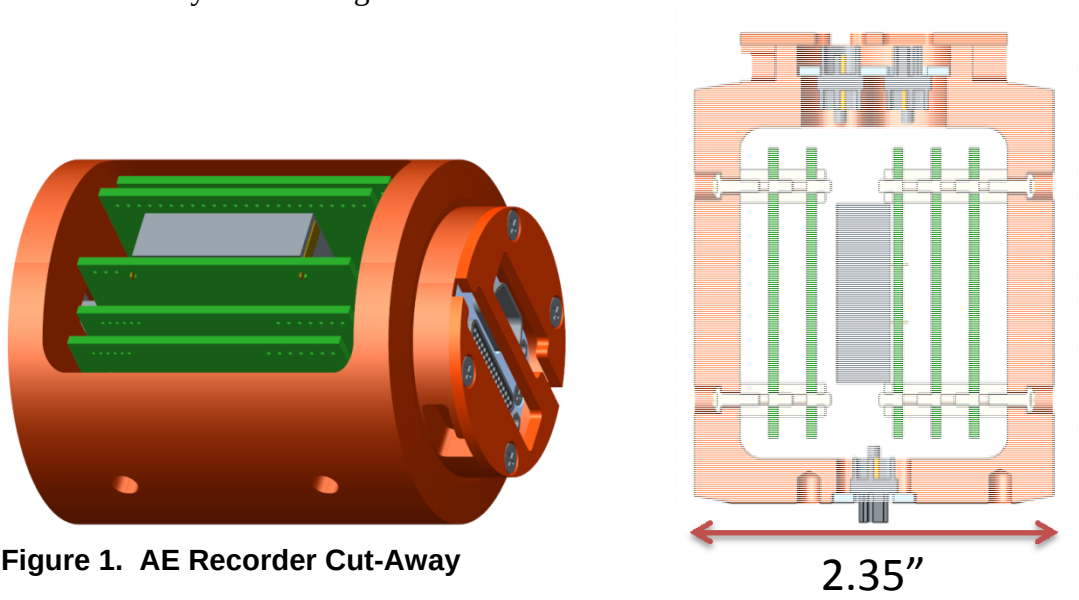


Figure 1. AE Recorder Cut-Away

Design Requirements

Requirements for the AE Recorder were negotiated with the impact test customer. Each of the 12 accelerometer channels has electronically programmable gain and balance. The remaining two analog channels measure the internal capattery voltage and external battery voltage, which are useful for validating data and planning future tests. Although no channels are specifically dedicated for use as a check, or dummy, channel, customers are strongly urged to allocate a channel for this purpose to allow verification that the data collected were not corrupted by the instrumentation itself (see the discussion in section Why a Check Channel Is Important). Using a check channel is very important because the system is intended for use in new regimes beyond current levels of impact shock where instrumentation capabilities are also unproven. The performance of the AE Recorder as implemented is summarized in Table 1 below.

[1] During preparation for a rocket sled track event on 9 October 2008, there was an unexpected ignition of the Zuni rocket motor. Three Sandia staff and a contractor were involved in the accident; the contractor was seriously injured and made full recovery. The data recorder battery energized the low energy initiator in the rocket.

Table 1. AE Recorder Features

User Channels:	
Accelerometer Channels:	12
Housekeeping:	2 – Battery Voltage, Capattery Voltage
Discrete (Bi-Level)	8 – Trigger Fiducial, Sync, and 6 comparator
Sample rate:	250k samples per second per channel
Anti-alias low-pass filter:	7-pole Butterworth, 50kHz bandwidth
Recording Time:	213 seconds maximum with 75ms pre-trigger
Gain, reprogrammable:	6 to 320, steps of 6.25 x (1 .. 128)
Balance, reprogrammable:	0 to 5V

Trigger Configuration	
Trigger channel:	Any analog or bi-level channel
Trigger window:	Programmable 0 to 100% of channel range
Trigger qualification width:	Programmable 1 to 255 samples (3.4ms)
Trigger modes:	Internal trigger channel, Computer trigger
Arm modes:	Computer Arm, Delay Arm, Arm on Power
Arm delay time:	0 to 1023 minutes

Power and Control	
Supply voltage:	16V maximum, 6V minimum
Supply current:	Low-power (delay arm): 25mA @ 16V; Full-power: 150mA @ 16V
Internal capattery:	16V, 120mF providing 5-sec nominal power
Baud rate:	115.2k Baud command, 150Mbps SerDes data

Environmental and Mechanical	
Operational shock:	More than 20,000 G
Temperature:	-20°C to 70°C
Data recorder weight:	1.4 pounds
Materials:	PH 13-8 Mo Stainless Steel
Encapsulation:	828/DEA/GMB with procedure SS2A0780
Data recorder dimensions:	2.35” diameter, 3.0” tall

Other requirements are internal stored energy that allows the recorder to operate after external power is lost; the ability to self-monitor the impact environment once the recorder is armed, and then self-trigger; signal processing of accelerometer signals with a low-pass filter; and obviously for a memory-based system the need to survive impact and retain the data collected for later retrieval. Although not required for the initial application, a stand-alone operating capability facilitates gun-launched tests so that only an external battery is needed, and leads to broader applicability of the AE Recorder for other high-shock measurement applications.

Design Implementation Summary

The internal stored energy requirement is fulfilled using a special robust component called a double-layer capacitor or capattery, which can supply power for 5 seconds following battery loss. During the 3DDR-AM development on which the AE Recorder was based, we investigated concerns of mechanical reliability for this component with laboratory tests plus modeling and simulation (3). Data are stored in two types of non-volatile memory, so do not require keep-alive power post-test to hold information.

Self-triggering when in an armed mode is especially useful for gun-launched penetrator tests. Any single or multiple analog or discrete channel can be configured as the recording trigger. The recorder is normally configured to trigger on any significant acceleration change, so for gun tests the acceleration impulse is also be recorded, so that the complete launch-to-impact profile is captured. Self-triggering also minimizes external wiring since the recorder can be left in arm mode for hours while waiting for trigger, with the time limited only by battery life.

Because accelerometers are the most common sensor used with shock-hardened recorders like the AE Recorder, signal processing is a necessity for removing high-frequency signal components. No real-world filter has a “brick wall” abrupt filter characteristic, so the corner frequency must be selected to consider both the sample rate and realistic filter characteristics. Even at the high 250k samples per second offered by the AE Recorder, aliasing will result unless a low-pass filter attenuates the contribution of frequencies beginning at about 50 kHz.

In addition to their wide output bandwidth, another characteristic of piezoresistive accelerometers is non-zero offset that varies with the device and the same device over temperature. For this reason, the AE Recorder includes a field-adjustable offset adjustment. Even with the signal balanced when the unit is first armed and while waiting for trigger, temperature changes will result in a signal shift of unpredictable magnitude and direction. The AE Recorder includes a patented algorithm for continually adjusting the trigger threshold to compensate. (4)

Field-programmable gain allows adjusting the signal range of interest to meet the expected sensor output. The settings are in convenient binary increments: 1, 2, 4 ... 128 times the base gain. The initial version of the AE Recorder used a 1.25 base gain, and the second version used 6.25. This range covers nominal outputs for sensors appropriately selected for the expected impact environment.

The threshold for the six user discrete signal inputs is also adjustable in the field from near zero to nearly 3V. Two additional discrete signals are internal and use TTL-level thresholds: Trigger which marks detection of the trigger fiducial, and Synchronization which is intended to capture a common signal applied to a group of recorders that allows later alignment of the recorded data post-test.

An important concern in a recorder design with a Gigabyte of memory is time to extract the data. This was addressed on the AE Recorder by developing a high-speed, serial interface based on the SerDes (Serializer-Deserializer) electronic device. The 150Mbps rate enables data extraction at about the same speed it is recorded. A second low-speed interface at 115.2k Baud permits

recorder configuration and can be used for data extraction. We developed special fielding interface hardware to encompass both interface types in a simple, USB-connected interface.

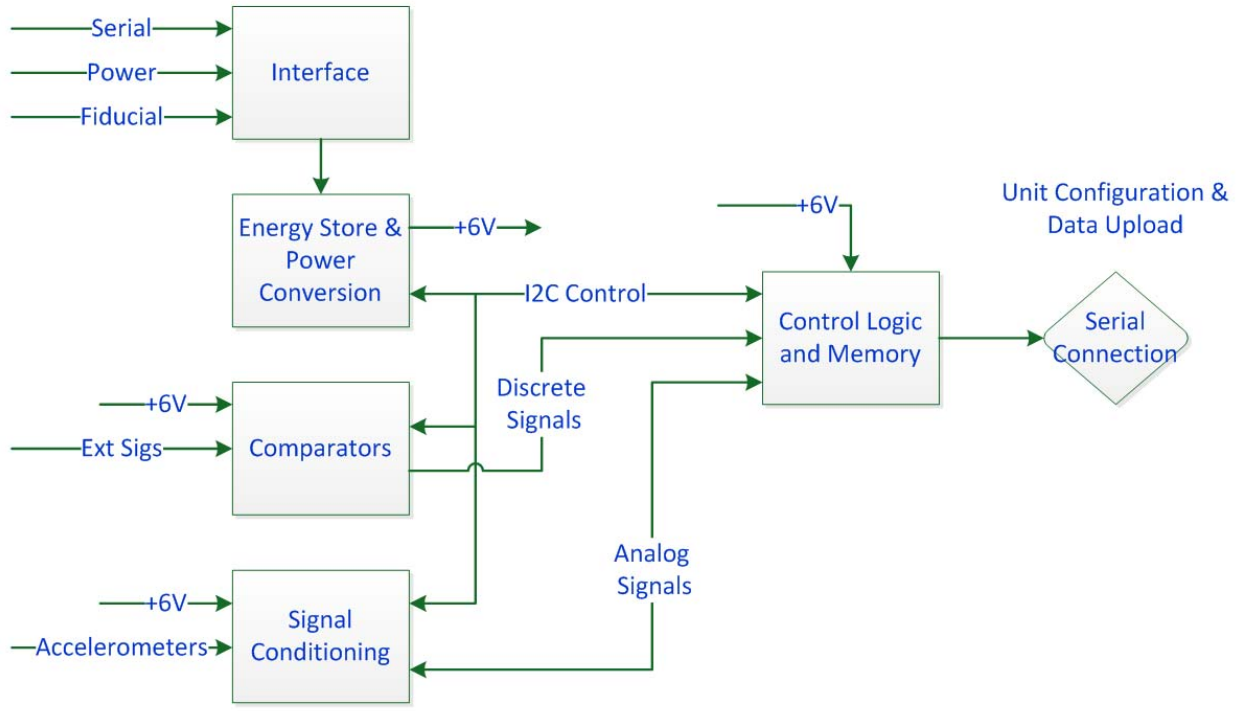


Figure 2. Block Diagram of the Anomalous Environment Data Recorder

The electronics design block diagram, as shown in Figure 2, was segmented into three circuit board types, resulting in 5 circuit boards in the AE Recorder housing because the analog / signal conditioning board is used three times with four channels each. The "Interface" and "Energy Store & Power Conversion" blocks are implemented as one circuit board, 3A6999-002 Power / Interface. A switch-mode power converter on this board reduces the capattery voltage to 6V for distribution to all other boards. The "Comparators" and "Signal Conditioning" blocks, implemented on the 3A7000-002 Analog board, accept external signals and condition them for digitization on the 3A6998-002 Digital board. The Digital board implements the "Control Logic and Memory" block, with logic in the Actel Smart Fusion gate array directing the output from the two, 6-channel Analog-to-Digital Converter chips using into either the 75 millisecond deep FRAM non-volatile circular buffer or the 210-second deep NAND Flash. The "Serial Connection" interface for configuration, command, and high-speed data extraction is also implemented on the Digital board. The concept of microcontroller-plus-FPGA logic was first employed in HiCapPen (5), but improved here with the use of a single device combining microcontroller and logic functions.

Auxiliary Equipment for Fielding

Although the AE Recorder forms the data acquisition system core, varying levels of other support electronics are needed to complete the instrumentation depending on the application. This auxiliary equipment can be as simple as a battery for a gun-launched test, but the other

extreme is a sled track test with electrical isolation that requires fiber-optic communication and internal battery power with a built-in battery management system.

Fiber-Optic Communication and Control

For the B61 sled track test, the AE Recorder on-board instrumentation uses a multi-mode fiber-optic connection to control power on and exchange data. At the sled track facility, single-mode fiber from the track control room runs more than a mile to a track-side box near the test article. At that location, the single-mode fiber is converted to multi-mode for connection to the test article using an assembly of commercial fiber media converter modules. The multi-mode fiber runs as far as 1000 feet from this point to a fiber junction box near the test article. The last 20 feet of fiber from that junction is terminated in a multi-fiber connector, which finally connects to the test article.

Using multi-mode fiber on the test article was a practical adaptation to the extremely dusty outdoor environment of sled track operations, where dust particle size approximates the fiber diameter of single-mode fiber. On board the test article, multi-mode fibers are used to drive photo-diodes to turn system power on, and transceivers to exchange command and configuration data. After qualifying the turn-on signal, the 3A7002-003 External Fiber-Optic Interface Board latches power and waits for commands. The microprocessor monitors both receive channels, and if the primary channel fails the input is switched to the secondary input fiber. The selected serial stream is applied to all the recorders, which then select messages based on address. The fiber-optic transmit always sends on both fibers simultaneously.

On-Board Instrumentation Power Source

In addition to the fiber interface, the 3A7002-003 External Fiber-Optic Interface Board contains switch-mode power converters to reduce the 36.5V maximum $LiFePO_4$ battery voltage to 16V for distribution to the AE Recorder modules, and another SMPC to step down the 16V to 5V for use by the fiber optics and microprocessor. Each AE Recorder module draws about 150mA at 16V with accelerometers connected. The 3A7002 board, and the 3A7004 Battery Management System board which receives 5V power from the 3A7002 board, draw about 50mA at 16V. Total current for the four AE Recorder modules and support equipment is then about 650mA at 16V. The SMPC producing 16V output from the 32V nominal battery voltage is about 80% efficient, so draws less than 0.5A from the battery which is composed of 10 each K2 Energy LFP26650P80 cells, 2.6A-Hr nominal. Thus, a fully-charged battery should operate the system for about 5 hours. During non-operational time, the battery management system is set to Sleep mode which draws about 0.1mA, equivalent to less than 5% self-discharge after one month.

An internal, rechargeable battery seemed the best design choice for providing instrumentation power. Using an externally-attached battery is not feasible when explosives safety is considered because a battery provides energy compatible with explosive initiators. With the internal battery, mitigation was needed to ensure the battery remained isolated from electrical conductors penetrating the test article's metal surface. Following the first sled test, an alternative was raised that applied external, remotely-controlled power just before launch to a large, internal capacitor bank. This idea was evaluated but not pursued due to time and funding constraints.

Once established that an internal battery would be used, it seemed prudent to use a rechargeable battery. Although the internal battery could have been a primary (non-rechargeable) battery, on-board instrumentation assembly and check-out can use varying and unpredictable amounts of power. Time between test article assembly and the actual test date could also be several months. Partial disassembly of the test article to replace an internal, isolated battery was unacceptable. Thus, a secondary (rechargeable) battery was selected using $LiFePO_4$ chemistry. The very low self-discharge rate from $LiFePO_4$ and the associated battery management system should allow recharging after test article assembly and check-out is complete and no further servicing for several months, however, since the recharging capability is readily available, the battery is topped off on the day of test.

Electrical Connections for Test Article Status and Battery Charge

Finally, 3A7001-002 Safe-State Monitor interface panel contains electrical connections to monitor and service the battery, LED indicators that instrumentation power was active, and a location from which to distribute flashing "Blink" LEDs to correlate test article movements captured on camera with the AE Recorder data. The electrical connections are in a socket-type circular military connector covered by a metal dust cap to ensure no exposure of electrical conductors.

The same circular military connector is used on the 4A1390 Test Article Interface Box, which provides a break-out connections for the test object's circular connector signals that are used for voltage verification, isolation tests, and battery charging. The box is disconnected from the test article before launch. The box also provides a laptop computer-to-fiber optic connection to the test article. The test object's fiber-optic connector is located away from the interface panel. The user interface software loaded on the test track console room computer is installed on the laptop, which along with the interface box, translates fiber-optic controls for use during test article assembly and check out.

One more item is helpful for procedure development and practice using the sled track configuration. A 4A1396 Test Article Stand-In enables test track personnel to develop and thoroughly practice procedures. It contains copies of the same circuit boards used in the real on-board instrumentation, so reacts identically to the test article. It is designed to interface using the 4A1390 Test Article Interface Box.

Why a Check Channel Is Important

All electronic components have varying degrees of piezoelectric response, but in the AE Recorder design this is practically limited to ceramic capacitors constructed of ferroelectric dielectrics such as X7R. Research during the HiCapPen development (5) provided design guidance that only C0G-type ceramic or tantalum capacitors should be used in the analog amplification and signal conditioning circuits. Common mode noise injection may also occur outside the analog signal chain from sensor excitation and analog circuit supply voltages. For the bridge-type piezoresistive accelerometers, common-mode rejection is a function of the bridge quiescent imbalance and the bridge resistance itself, driven by noise on the excitation voltage.

Unanticipated piezoelectric problems can occur in spite of this understanding of the phenomena and using correct design principles. For this reason, users are strongly encouraged to dedicate

some channels to check channels, also called dummy channels. When piezoelectric contamination occurs, it is correlated with the expected acceleration measurements. The data output plot in Figure 3 was taken from a gas gun launch pulse on an old development system called AdPen-NV. This check channel had a dummy accelerometer input, but the output clearly was not zero. Because the contamination is correlated with the intended acceleration, common validation methods that integrate the acceleration measurement to produce velocity and displacement data fail to detect the corruption.

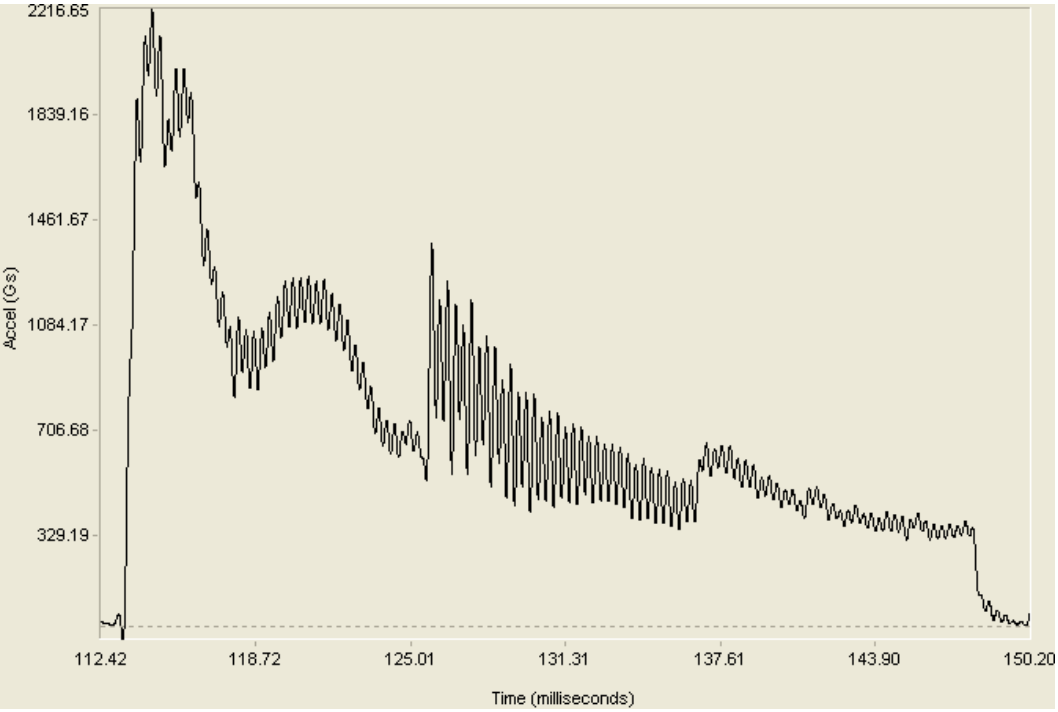


Figure 3. AdPen-NV Check Channel Showing Piezoelectric Data Contamination

Project Costs and Duration

When the AE Recorder project was first funded 20 November 2014, a $910k development cost was estimated. Requirements discovery early in the project revealed the additional requirement to electrically isolate the instrumentation from anything contacting or outside the test article. The subsequent design and implementation of an electrically isolated fiber-optic system to control the test article from the sled track control room cost by itself $445k. Additional incidental development affected the schedule and cost as well, such as design changes and engineering work-arounds in response to Critical Design Review action items late in the development. When the system was delivered December 2015, costs totaled $1.7M, with $1.4M labor and $250k purchases.

Following the completely successful March 2016 sled track test, the customer requested additional units but with design modifications. Minor changes affected nearly all the previously designed circuit boards. Improvements in this build also included a stand-in test article with greater similarity to the actual unit, more robust electrical connections for verifying isolation of the instrumentation, and support hardware to simplify test article assembly and check-out. Acknowledging that incremental costs are lower to build two sets of instrumentation now instead

of delaying that acquisition, we built two sets of on-board instrumentation to support the upcoming sled track test plus a subsequent test. The $641k project funding early June 2016 was anticipated to provide delivery November 2016.

MECHANICAL DESIGN

Sandia has developed earth penetrator instrumentation since 1974. In the beginning, the large, 6” diameter mechanical packages reflected the electronic technology at the time, with Dual-Inline-Package (DIP) integrated circuits and very low levels of device integration. The first microprocessor-controlled penetrator instrumentation, AdPen, was developed in 1996. Higher levels of device integration and smaller component packaging resulted in a merely 3” diameter recorder about 9” long with the first appearance of a rectangular notch in the steel cylinder supporting the electronics. This same concept appeared in all subsequent penetrator instrumentation designs and is successful because of its mechanical strength and relatively simple electronics assembly process.

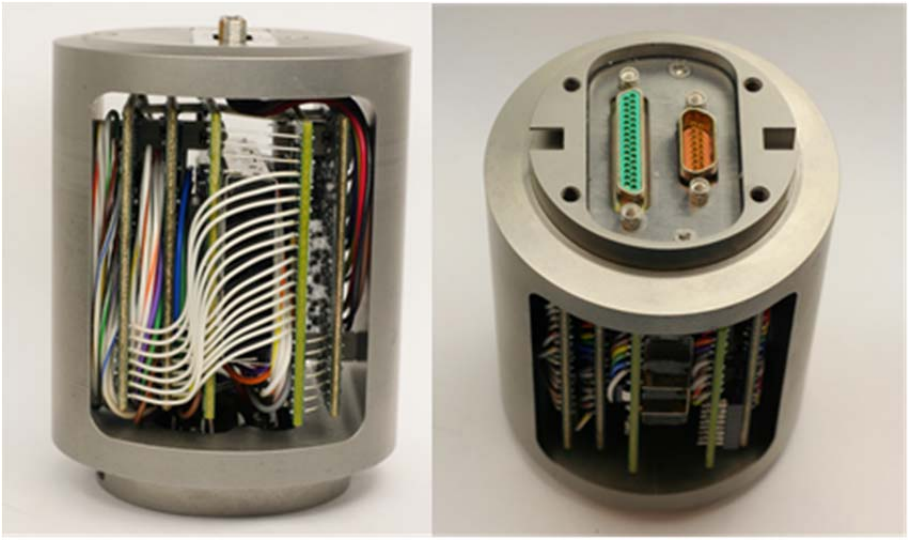


Figure 4. AE Recorder Assembly before Encapsulation

Development of the AE Recorder housing relied heavily on mechanical modeling and simulation. Design trade-offs within the mechanical constraints led to parsing the electronic design into five circuit boards of three types: Digital, Power/Interface, and three copies of the Analog board. The Digital and Power/Interface boards required the most area, and consequently they were located near the cylinder center. Board widths included sufficient space to accommodate routing connecting wires over the board edges and yet leave sufficient encapsulation depth, as shown in Figure 4. A benefit of the unconstrained, stirrup housing is that it allows the epoxy-based encapsulant to relax during the curing process and relocate maximum principal stresses away from the electronics.

Harnesses from an MDM-31P and MDM-15S on top and MDM-15P on bottom route to through-hole connections on the boards. The plug-type connectors service the Analog boards, with 15 wires allocated to each board. This means the MDM-15P harness on the bottom only connects to Analog Board 1, while wires are bifurcated from the MDM-31P to Analog 2 and Analog 3.

About half the MDM-15S harness goes to each of the Digital and Power / Interface boards, with all serial communication to the Digital board and power and discrete control to the Power / Interface board.

The boards are mounted within the assembly using ten 3/16" length, #2-56 nylon standoffs to mechanically suspend the boards within the housing. Two removable panels on top and bottom of the recorder allow the boards to be assembled independent of the housing; once assembly is complete, the electronics are inserted into the housing and mounted in place. While this feature successfully prevents technicians from having to build a "ship in a bottle", this is by no means an easy product to assemble. Future revisions, if pursued, should focus on manufacturing improvements. At the very least, pre-cut and formed ribbon wires or flex cables should be designed to reduce the number of individual wire routes in the assembly.

For the B61-12 sled track application, each recorder was housed in steel sleeve to protect the exposed windows of the recorder. However, the top of the recorder (i.e., the side with two connectors), was still exposed. As a result, an aluminum cap was bolted to the top of the recorder as an added measure of protection for the connectors[2] (see Figure 5). Two small notches in the top of the housing of the recorder (also visible in Figure 5) can be used to remove the recorder from an upper level assembly such as a penetrator. Full details on the mechanical design can be found in the assembly drawing: 3A5279D01.

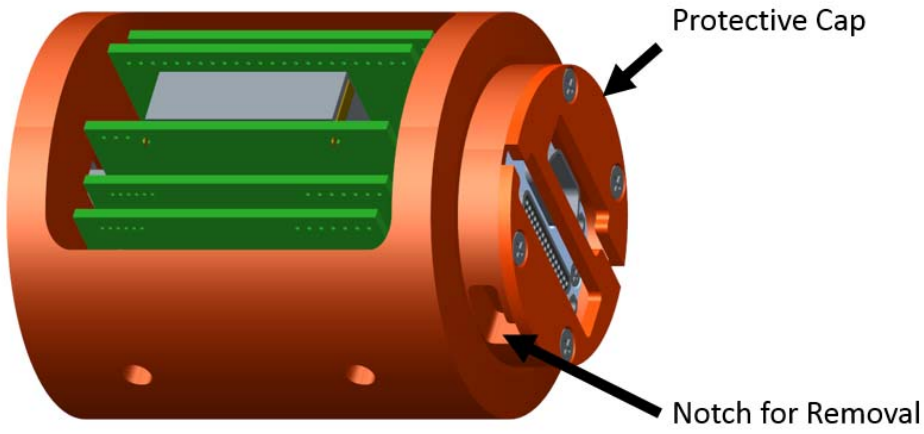


Figure 5. AE Recorder with Cap Installed.

Mechanical Modeling & Simulation

Science-based engineering that includes modeling and simulation increases the likelihood that a design is robust. A mechanical model is developed during negotiation with other team members and evaluated for flaws. This first stage establishes the physical interfaces to which the assembly will join, and the constraints from the electronic designer's objectives. Once the model has matured to a suitable point, the model's stress profile over the entire life cycle is then captured using a Multi-Stage FEA procedure. This is often an iterative process, where the mechanical

[2] Examination of the recorder hardware after the sled track test indicated that debris did indeed hit the top of the recorders. While some of the connectors were damaged, the cap did prevent direct impact to the potted electronics.

model is evaluated and tweaked until the final solution is determined. Although all real-world effects cannot be anticipated or modeled, this approach greatly improves performance during the product's life cycle.

Modeling and simulation was particularly important to the AE Recorder development because the recorder will be subjected to both harsh thermal and dynamic environments. While gun-launch and impact profiles for earth penetrator tests are known to some extent, the impact during a sled test was conjecture with many assumptions.

An important consideration for encapsulated systems is residual stresses from the encapsulation curing process. The elevated, 70°C curing temperature leaves thermally-induced stresses when materials with disparate coefficients of thermal expansion are used. Residual manufacturing stresses may significantly reduce material failure strength margins.

Another aspect is pre-loading. All gun-launched penetrator systems apply compressive loads of up to 5000 pounds to the instrumentation package when building the test article. This pre-load is intended to overcome compression applied to the recorder body on target impact, and avoids "chattering" of the steel surfaces that affect data quality and damage accelerometers.

Simulating both the residual stresses and pre-load are accomplished with Implicit Quasi-Static, Sierra S/M: Adagio. The simulation run shows stresses as the object is cooled from encapsulation cure temperature down to the lowest operational temperature (in the case of the sled track, -20°C). Stress is assumed to be zero at the elevated cure temperature. The model is simplified by removing all electronic components from the boards except the large capattery. Then, the simulation applies the compressive pre-load to the AE Recorder's chamfered mounting surfaces.

Finally, the anticipated impact acceleration is simulated using Explicit Dynamic, Sierra S/M: Presto at the lowest operating temperature. As before, the forces are applied to the model's chamfered surfaces.

In summary, the FEA procedure is listed below:

1. Cool from potting cure temperature of 70C, to lowest operational temperature of -20°C (Quasi-static, S/M: Adagio)
2. Apply 5000 lb. preload to chamfers (Quasi-static, S/M: Adagio)
3. At -20°C, accelerate using chamfered load surfaces with 25kG, 0.5ms haversine pulse (Dynamic, S/M Presto)

The mesh of the finite element model is shown in Figure 6. The mesh contains 811,132 Hex8 Elements with an average size of 0.020". Half symmetry was used to reduce computation time/cost. As previously mentioned, individual electrical components were not modeled.

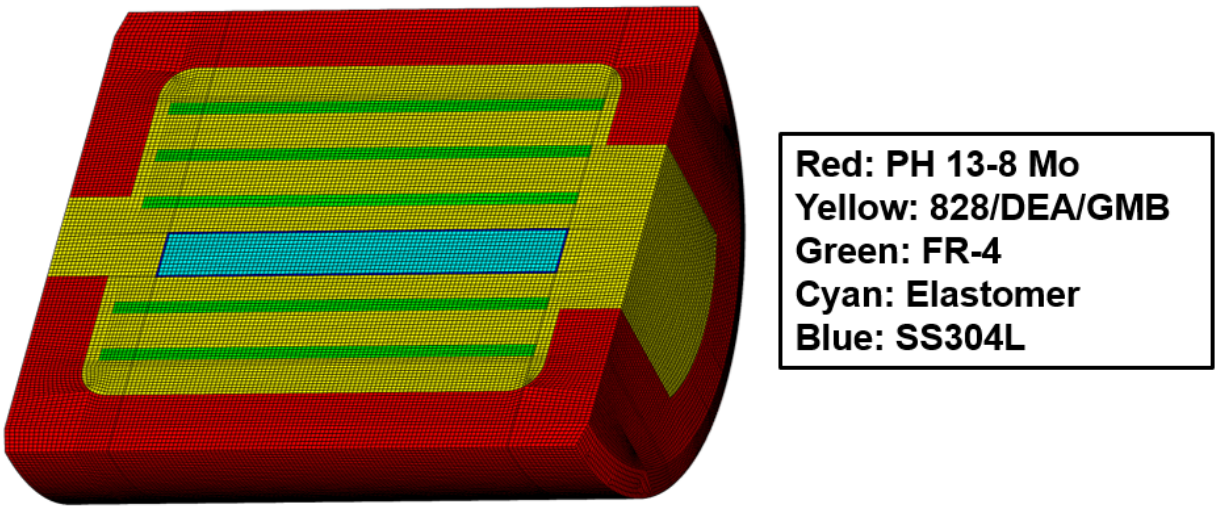


Figure 6. AE Recorder finite element model mesh.

The housing material was modeled using an elastic-plastic power law hardening model for PH 13-8 Mo stainless steel in the H950 condition. The shell of the capattery was modeled using a thermo-elastic plastic power law hardening model for 304L stainless steel. The power law parameters for both of these materials can be found in Table 2, and the temperature dependent functions for the 304L can be found in Table 3. The 828/DEA/GMB potting material was modeled using the simplified potential energy clock model with parameters defined in Reference (6). Likewise, the stiffness matrix used to determine input parameters for the 3D orthotropic elastic model of the FR-4 material can be found in Reference (6). The elastomeric material used to model the "guts" of the capattery were assumed to be the following: G = 0.26MPa, K = 10MPa, $CTE_{transverse}$ = 31 ppm/C and CTE_{normal} = 84 ppm/C.

Table 2. Elastic Plastic Power Law Hardening Model Parameters for 304L and PH 13-8 Mo Stainless Steels.

Parameter	PH 13-8 Mo	304L
Density (kg/m^3)	7800	7920
Young's Modulus (Pa)	175E+9	194.5E+9
Poisson's Ratio	0.264	0.264
Yield Stress (Pa)	800E+6	206.8E+6
Hardening Constant (Pa)	1.0E+9	0.86464E+9
Hardening Exponent	0.1	0.53574
Luder's Strain	0.0	0.0
Beta	1.0	1.0

Table 3: Temperature Dependent Functions for Thermo-Elastic Plastic Power Law Hardening Model for 304L Stainless.

Temperature (K)	Thermal Strain function	Young's Function	Poisson's Function	Yield Function
218	-0.001264	1.003	0.9697	1.1667
293	0.0	1.0	1.0	1.0
373	0.001348	0.97766	1.0303	0.8233
473	0.003107	0.9464	1.0606	0.69
573	0.004934	0.9085	1.0909	0.6
673	0.006840	0.8656	1.1174	0.53
773	0.008832	0.8177	1.1439	0.47
873	0.010846	0.7667	1.1704	0.42

The FEA results for the housing are listed in Table 4. A contour plot of Von Mises stress in the housing at 25kG peak acceleration is shown in Figure 7. The maximum stress in the housing is seen at the corners of the windows. The peak Von Mises stress at -20°C and with the 5000 lb. preload is 657 MPa, which gives a safety factor of 2.15 to the yield stress of the housing material (PH 13-8 Mo, H950). At 25kG (and at -20°C with 5000 lb. preload), the maximum Von Mises stress in the housing increases to 849 MPa, and correspondingly reduces the safety factor to 1.67. With these boundary condition assumptions, the housing is not expected to yield.

Table 4: FEA Results for AE Recorder Housing.

State	Max Von Mises	Safety Factor*
-20C with 5000 lb. Preload	657 MPa (95.3 ksi)	2.15
During 25kG, 0.5ms Pulse	849 MPa (126 ksi)	1.67

* Safety Factor based on minimum yield strength of PH 13-8 Mo in condition H950 – 1415 MPa (205 ksi), per ASTM A564.

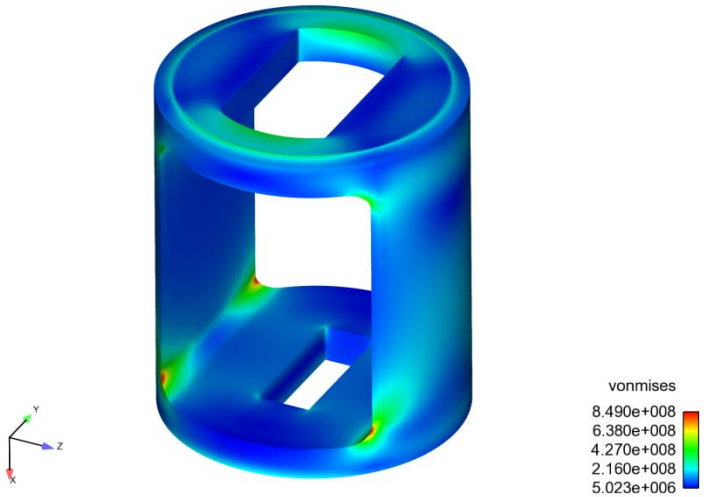


Figure 7. Housing Von Mises Stress State at 25kG Peak (Full Symmetry Shown.

The FEA results for the potting are listed in Table 5. A contour plot of the maximum principal stress in the potting at 25kG peak acceleration is shown in Figure 8. The peak maximum principal stress in the potting is seen at the corners of the windows where the potting interfaces with the housing. The peak maximum principal stress at -20°C and with the 5000 lb. preload is 53 MPa, which gives a safety factor of 1.32 to the yield stress at room temperature of the potting material (828/DEA/GMB). At 25kG the peak maximum principal stress in the potting increases to 68 MPa, and correspondingly reduces the safety factor to 1.03. While the peak stress is high and may indicate that adhesive failure will occur at the interface of the housing and the potting in the corner of the window, this location is away from the electronics and is not expected to affect the performance of the recorder. In fact, the stress state near the electronics is very low.

Table 5: FEA Results for AE Recorder 828/DEA/GMB Potting.

State	Max, Max Principal Stress	Safety Factor*
-20C with 5000 lb. Preload	53 MPa (7.69 ksi)	1.32
During 25kG, 0.5ms Pulse	68 MPa (9.86 ksi)	1.03

* Safety Factor based on yield strength of 828/DEA/GMB – 70 MPa (10 ksi).

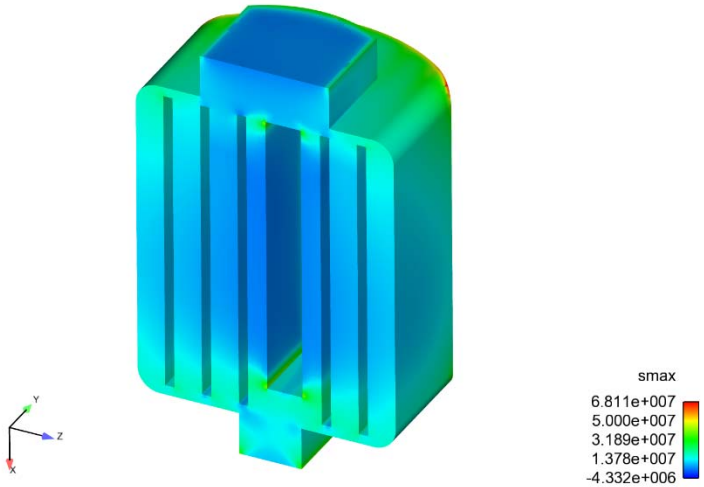


Figure 8. 828/DEA/GMB Potting Maximum Principal Stress State at 25kG Peak (Half symmetry shown.)

One item of interest is the capattery housing. The equivalent plastic strain of the capattery housing at -20C with the 5000 lb. preload applied is shown in Figure 9. As shown, the housing walls permanently deform outward due to the shrinkage in the potting material as the assembly cools from the elevated cure temperature down to the lowest operational temperature. During the 25kG peak, the equivalent plastic strain increases slightly from 0.04% to 0.05%. While these strains do not indicate that the housing itself is going to fail, the housing is expected to maintain electrical contact with the capattery core during operation, and therefore these plastic strains could cause the capattery to fail open. This same issue has been observed in other recorder designs (3DDR-AM and 3AMP); the solution has been to order special capatteries from AVX that contain conductive epoxy between the capattery core and the housing. For more information see Reference (3).

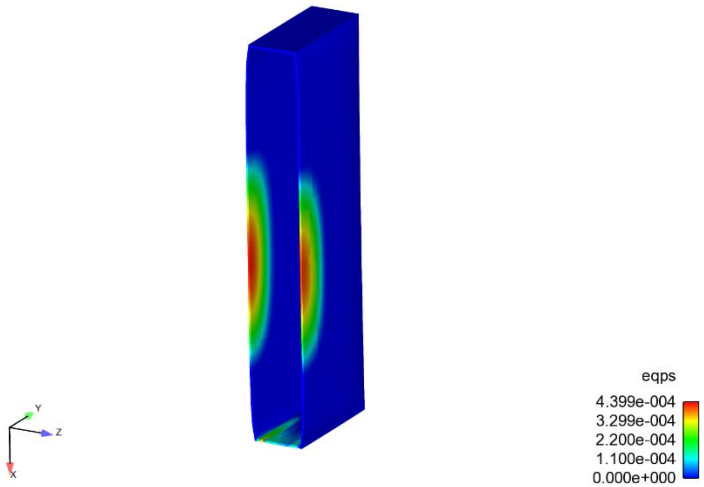


Figure 9. Capattery Housing Equivalent Plastic Strain at -20C with 5000 lb. Preload (Half symmetry shown at 10x displacement magnification.)

In conclusion, the FEA of the AE Recorder does not reveal any significant mechanical design flaws. The analysis was conservative in that it was performed at the lowest operational temperature and included loads and boundary conditions to account for the entire lifecycle of the product. The results illustrate the importance of including the manufacturing process in the analysis, as the residual stresses due to the cure schedule account for a significant portion of the final stress state in the AE Recorder.

Encapsulation

Research that modeled encapsulation systems over temperature and during shock revealed weaknesses in our formerly standard polysulfide rubber conformal coat / Hysol approach (6). Essentially, the conformal coat was ineffective in reducing thermally-induced stresses created during the 70°C epoxy cure cycle, and was dropped completely. The AE Recorder encapsulant scheme uses 828/DEA/GMB, plus component underfill using 828/D230, filled with 20% by volume alumina. The formulation for 828/DEA/GMB is diglycidyl ether of bisphenol A (Epon 828, Resolution Chemicals) cured with 12 PHR diethanolamine (DEA, Fisher Scientific) and filled with 48% by volume of Glass Micro-Balloons (D32/4500, 3M Corp.), cured at 70°C. PHR is an abbreviation for parts per hundred resin, a ratio by weight. The glass transition temperature Tg for 828/DEA/GMB is 81.4 ±1.6°C. The Glass Micro-Balloons part of the formulation helps to block crack propagation and improve the coefficient of thermal expansion for the material. Component underfill using 20% alumina-filled 828/D230 leads to higher fatigue life (7). A high-velocity penetrator test series into hardened targets of the 3AMP penetrator data recorder demonstrated the robustness of the encapsulation scheme.

Following underfill of all components using 20% alumina-filled 828/D230, we follow the neutron generator encapsulation procedure SS2A0780. The 828/DEA/GMB is very viscous and prone to voids. To compensate for this, the process uses an evacuation step typical of any encapsulation process, but then adds curing under 80psi pressure. The pressure pot, which can hold at most two AE Recorder mold fixtures, is placed in an oven with the following temperature profile:

1. Hold the molds at 25°C ± 10°C for 2 hours minimum and 24 hours maximum.

2. Raise the temperature linearly to 50°C ±6°C in 2 hours ±30 minutes.
3. Hold the cure temperature at 50°C ±6°C for 12 hours ±30 minutes.
4. Raise the temperature linearly to 71°C ±6°C in 5 hours ±30 minutes.
5. Hold the cure temperature at 71°C ±6°C for 5 hours ±30 minutes.
6. Ramp down to 25°C ±10°C in 30 minutes minimum and 24 hours maximum.

Pressure is held until after the pressure pot is returned to room temperature in the last step.

The mold fixture consists of two aluminum pieces that clamp together to hold the AE recorder module (see Figure 10). The mold is filled with epoxy using the two sprue holes in the top of the mold. At the completion of the cure process, the mold is separated using threaded holes and screws to "jack" the two halves of the mold apart. Any excess potting from the sprue holes is removed by turning the AE Recorder module on the lathe. The final encapsulated product can be seen in Figure 11, and more details about the mold itself can be found in the mechanical drawing: 3A5740D01.

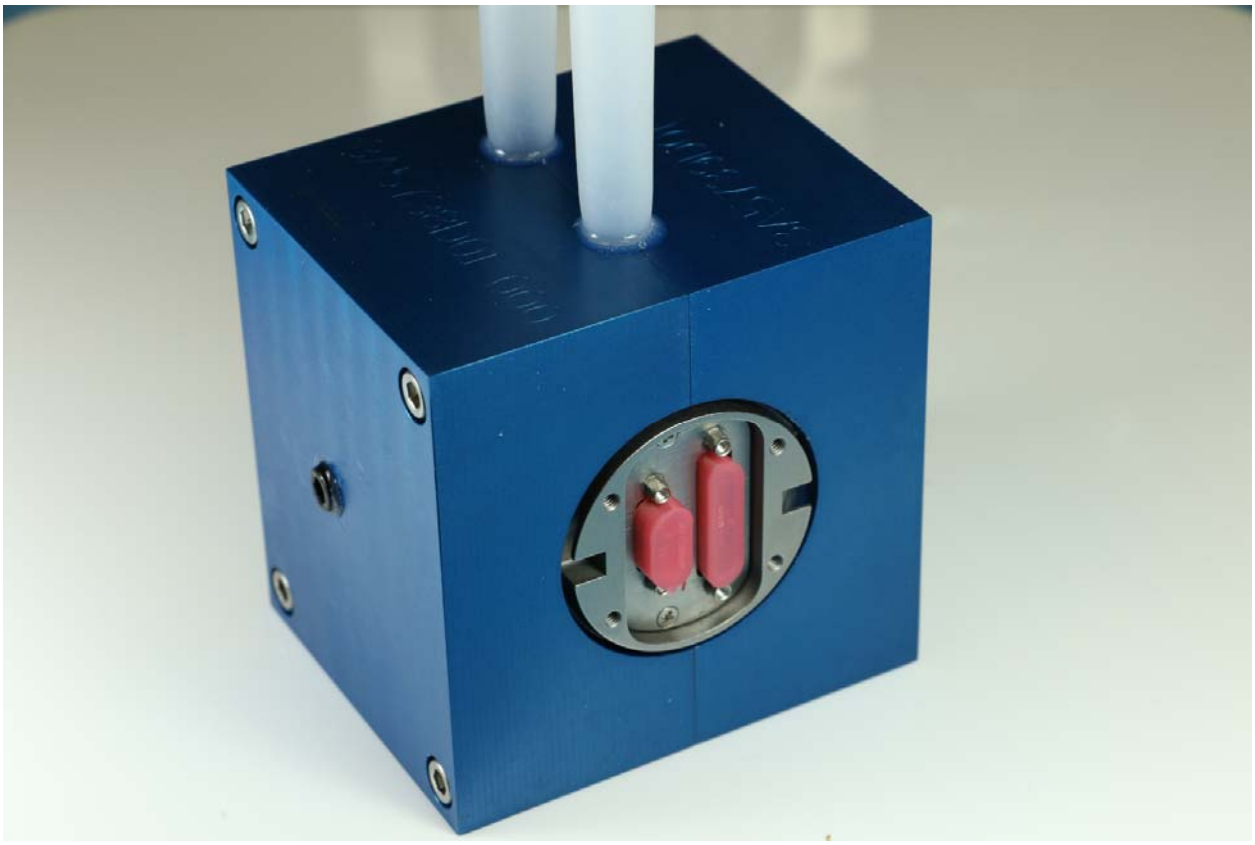


Figure 10. AE Recorder Mold Fixture Assembly.

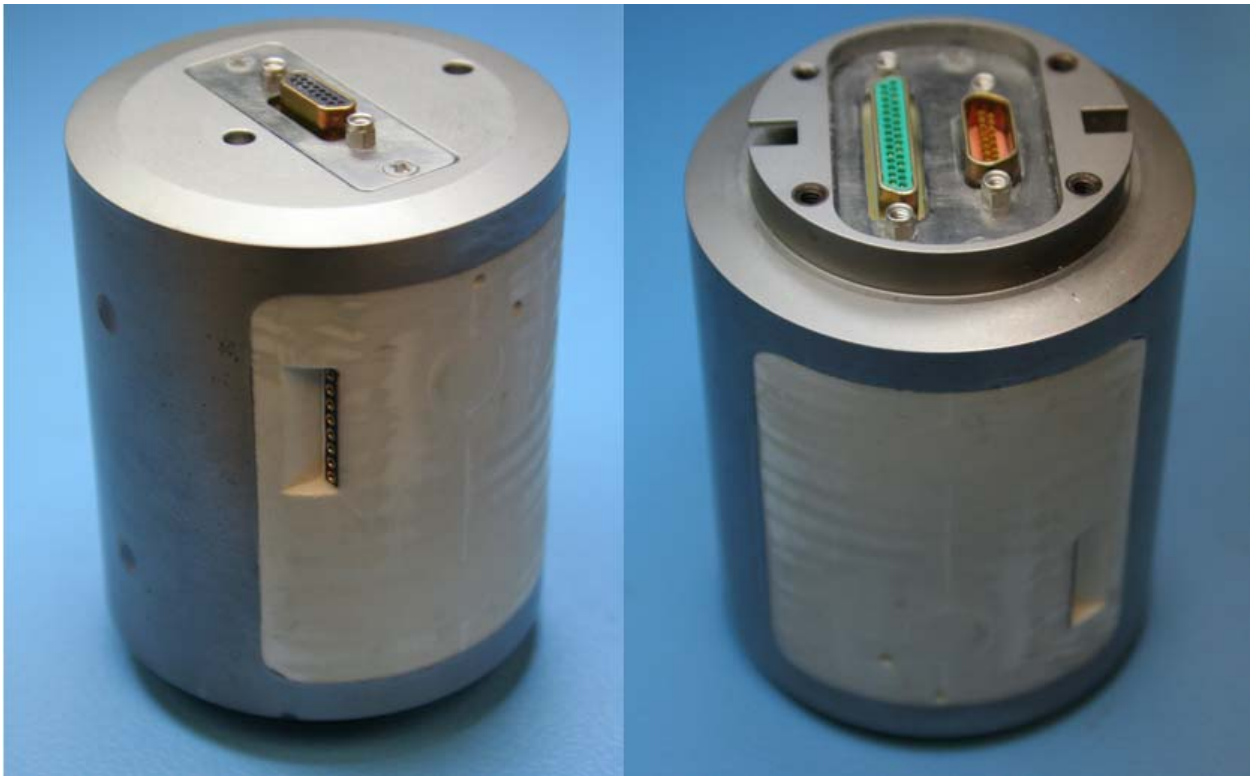


Figure 11. Encapsulated AE Recorder Modules.

ELECTRONICS DESIGN

Instrumentation on board test vehicles captures measurements used to support modeling and simulation of the impact conditions. Modelers wish for large number of channels and extremely high sample rates to support their analyses. Increasing sample rates are driven by interest in isolating smaller, component-level interactions of the test article. Instrumentation design for impact testing is a compromise among the number of channels, sample rates, power consumption, volume of the design, and survivability.

In response to the functionality needs and the constraints of the mechanical packaging, the AE Recorder design was parsed into three board types: Energy Storage & Power Conversion Power / Interface board; Signal Conditioning Analog board; and Control, Digitization, and Memory Digital board. The boards are linked with a common 6-wire bus containing 6V power and an I2C control interface. Commands from the Digital board configure the Power / Interface and Analog boards.

Energy Storage & Power Conversion

The instrumentation must be self-contained, without dependence on facility power. Batteries are the obvious choice for power but large capacitance could also be used with energy transferred before launch. The AE Recorder uses both a battery and a capacitor. High shock conditions at impact limit the reliance on batteries because they cannot survive impact conditions consistently even with shock-hardened packaging. But a battery is a convenient mechanism for charging a robust, internal energy storage element. For the shock-tolerant capacitor, the AE Recorder uses

an AVX BestCap capattery, also known as a double-layer capacitor or super-capacitor. The capattery uses a rubber-like polymer loaded with electrolyte that allows formation of a double-layer of ions. Because the separation distance between the ion layers is so small, very large capacitance can be created in a relatively small volume. The name capattery derives from the similarity to a battery with ion movement. However, the difference is the battery has a chemical oxidation-reduction reaction to store energy, and the capattery does not.

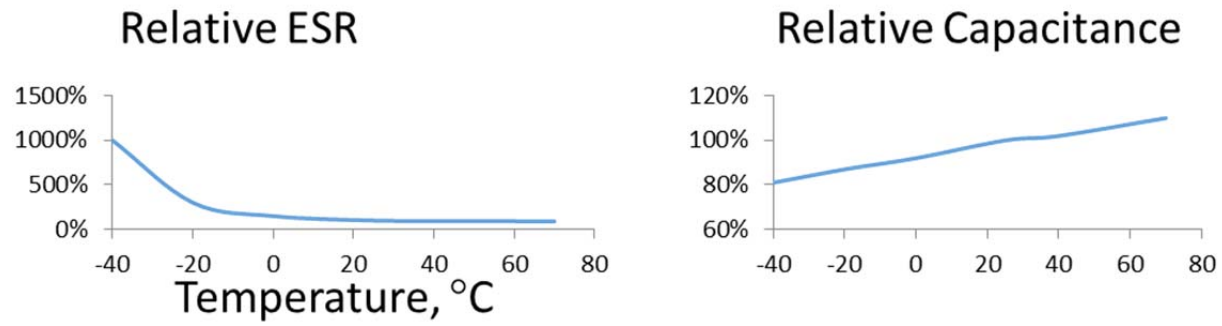


Figure 12. Capattery Temperature Effects on ESR and Capacitance

One important capattery characteristic is increased Equivalent Series Resistance (ESR) from 0.16-Ohms typical and reduced capacitance with reduced temperature as shown in Figure 12. At low temperature the increased series resistance produces a voltage drop when load is applied. This compounded problem of higher internal resistance and reduced capacitance is partially mitigated with switch-mode power conversion because the maximum energy is extracted from the capattery. Batteries have a similar cold-temperature problem, so the AE Recorder cannot be used much below -20°C.

Figure 13. Opened Capattery Shell with the Core Visible

The specific capattery in the AE Recorder design is the 16-V, 120-mF AVX BestCap BZ12GA124ZAB (8) shown in Figure 13 slightly disassembled to reveal the internal construction. The spot-weld bonds on the metal shell have been broken and the bottom shell, polymer core, and top shell, respectively, are slid apart. External dimensions of the shell are

approximately 1.9” X 1.16” X 0.33” and the device weighs about 24 grams. This model provides a higher energy density configuration than the standard AVX models. The design volume available limited the AE Recorder to a single capattery although multiple capattery units in parallel could improve reliability. According to the manufacturer a capattery is very unlikely to fail shorted, but from experience on other designs a capattery can fail open. This vulnerability is mitigated by a capattery special-ordered to use conductive epoxy between the core and the shell.

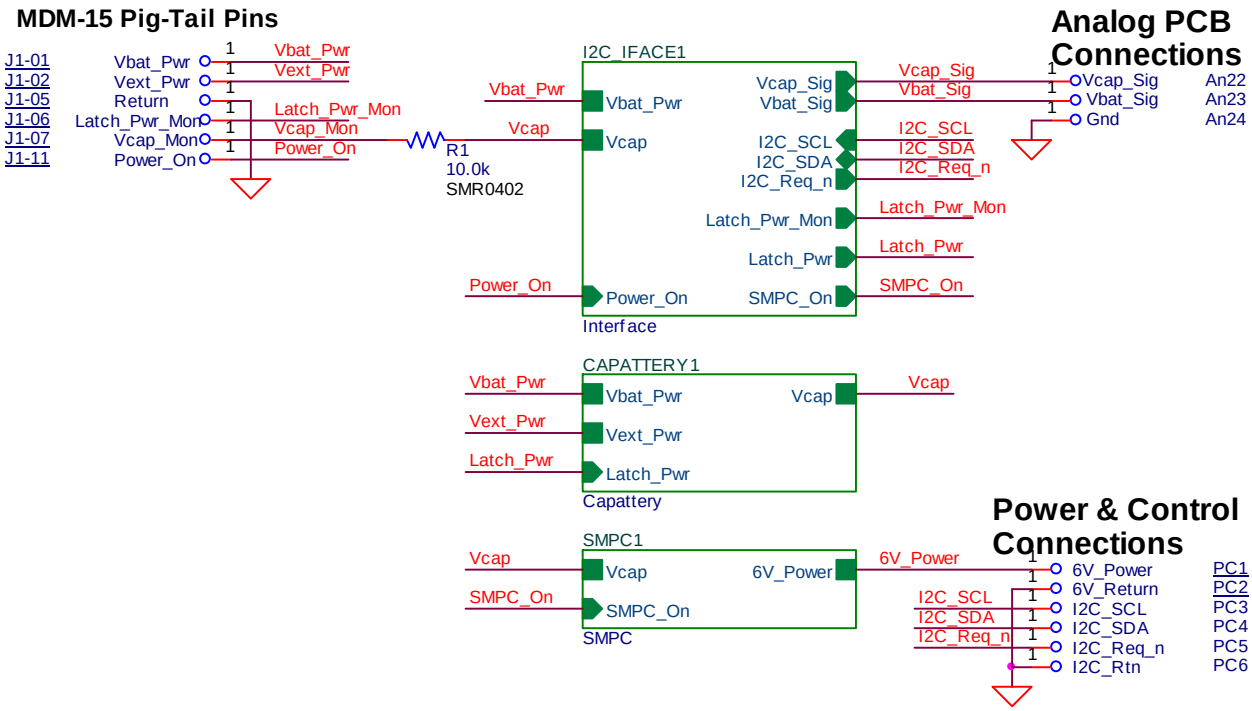


Figure 14. Top Level 3A6999-002 Power / Interface Board

To maximize both the capattery energy extracted and the resulting recording time, a switch-mode power converter (SMPC) steps down the 16V capattery voltage to 6V for distribution among the circuit boards. The Linear Technology LTM8031 ultra-low noise buck converter operates at about 85% efficiency with 16V input and 6V output. In addition to extending operating time on the capattery, using a SMPC reduces heat dissipation by more than 50% down to 2.4W from 5.4W if only linear regulators were used. Minimizing power is also important for extending battery life, and the SMPC allows a doubling of battery operating time. The hierarchical schematic top-level diagram for the 3A6999-002 Power / Interface board is shown in Figure 14.

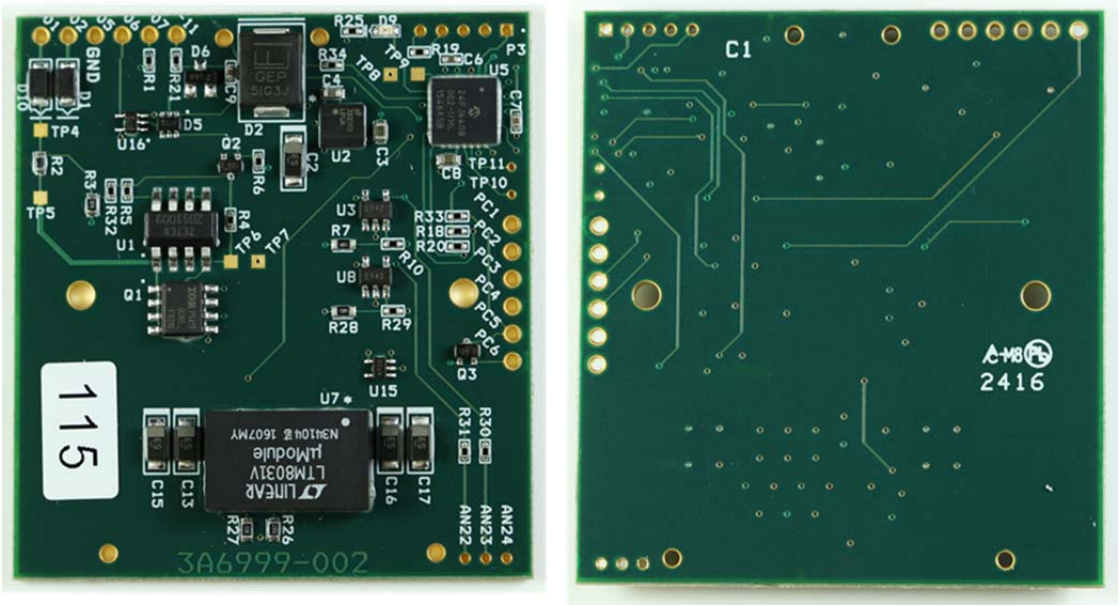


Figure 15. 3A6999-002 Power / Interface Board, Front and Back

The circuit board design as realized is shown in Figure 15. The capattery interface in the upper left includes diodes to apply Vbat and Vext power sources and the current-limit circuit on the Vbat input to clamp capattery in-rush current. All the external MDM-15 inputs have through-hole mounting along the upper left corner. The upper right quadrant contains the microcontroller and implements the "I2C_Iface" block in the top-level schematic from Figure 14. The microcontroller qualifies the Power_On pulse, latching power only after the signal is present at least 1 second, and enabling the SMPC. The lower third of the board contains the SMPC and filter capacitors. Board interconnections are along the center, right edge of the board, labeled "Power & Control Connections" in Figure 14. This bus is common to all five boards and includes the 6V power from the LTM8031 buck converter and the I2C control signals sent by the Smart Fusion microprocessor on the Digital board. The microprocessor processes the I2C address to respond only when addressed.

Each of the other board types include their own linear regulators to drop the 6V SMPC output that is bussed to all the boards to the voltages needed locally. Low-dropout regulators are used, which is particularly important for the 5V circuit voltage on the signal conditioning board and the 5V ADC supply on the digital board. For each of these linear regulators, we selected tantalum filter capacitors to minimize piezoelectric effects. If standard X7R dielectric capacitors were used, glitches could appear on the supply voltages at impact that would corrupt the collected data.

The 3A6999 board includes operational amplifiers to condition the Vbat and Vcap signals for measurement using the board's own microcontroller and the 3A6998 Digital board's FPGA. Measuring Vcap shows sufficient supply voltage was available to produce the 6V distributed to all the other boards and provides data validation. The battery voltage is useful for capturing when the battery is lost during the test and can inform instrumentation decisions on future tests.

Inrush current charging the large capacitance of the capattery needs to be limited, so we used a circuit that controls the gate voltage on a MOSFET based upon voltage across a current-sense resistor. Rather than tolerate the high voltage drop and accompanying power loss associated with driving the base-emitter voltage of a bipolar transistor with the sense resistor, we used a Zetek ZDS1009 (9) complementary current-mirror device to sense a much lower voltage across a smaller-value resistor as shown in Figure 16. Because the current-sense resistor can be a much lower resistance, power dissipation is decreased and a physically smaller, lower power rating resistor can be used resulting in smaller circuit volume. The permissible current is increased by decreasing the value of resistor R3. Decreasing the sense resistor R2 value to increase the current limit, as done in the classic circuit approach, would also work but is less flexible.

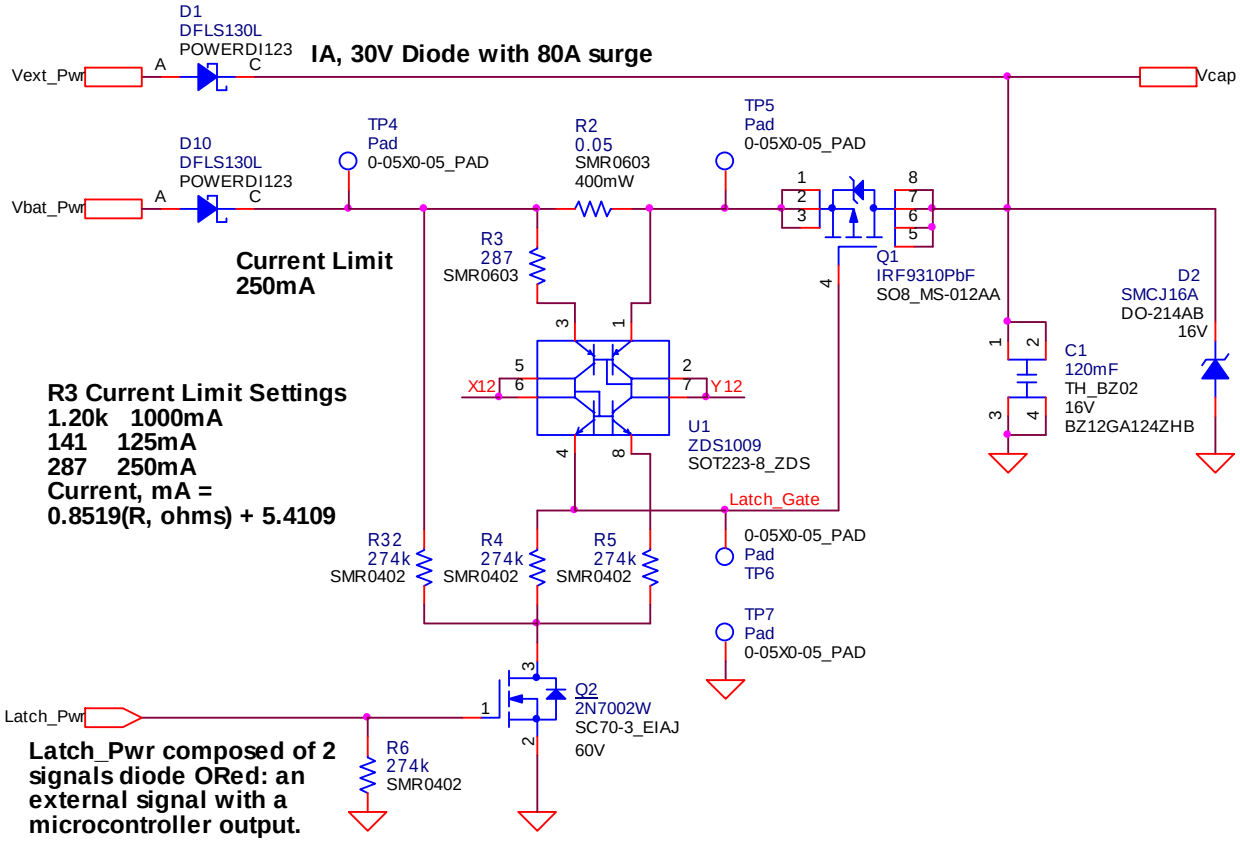


Figure 16. In-Rush Current Limit as Implemented Using a Current Mirror Circuit

The Microchip PIC24FJ64GB002 microcontroller has very limited functions on the board, primarily to validate that the Power_On pulse is at least 1 second before latching power to the capattery and then following an additional 8-second delay, to the Switch-Mode Power Converter. It also handles the I2C bus commands from the Digital board, with the 3A6999 Power /Interface board assigned address 0x10. Op Codes are shown in Table 6. Of particular importance is the Op Code to power the AE Recorder off.

Table 6. Power / Interface Board Configuration Op Codes

Op Code	Bytes	Function
Write, 0xAC	1	Latch power on
Write, 0x53	1	Unlatch power and turn system off
Read, 0x23	1	Report board power on cycles
Read, 0x25	1	Report number of times board has latched power
Read, 0x29	1	Report Vcap, capattery voltage
Read, 0x2B	1	Report Vbat, battery voltage
Read, 0x2D	1	Report temperature from MPC9701 sensor

Signal Conditioning

The AE Recorder is designed primarily to capture accelerometer data, the dominant sensor used in impact tests. Not all of an accelerometer's high-frequency signal content is useful, so must be sufficiently attenuated at the sampling frequency by a low-pass filter to avoid aliasing. Generic characteristics of ideal low-pass filter is shown in Figure 17. Aliasing, in which a high-frequency component appears as a lower-frequency signal, creates distortions not correctable with subsequent data processing. Accelerometer output frequencies are so high that amplifier slew rate limitations will cause non-linear effects and distortion unless a passive RC-filter precedes the first amplification stage. Because real-world filter implementations do not have abrupt cut-off frequency characteristics (the mythical "brick wall" filter profile), the gradual attenuation following the cut-off frequency must be accounted for in selecting the corner frequency and the sample rate. Phase compensation was included in the AE Recorder filter to avoid time correlation problems.

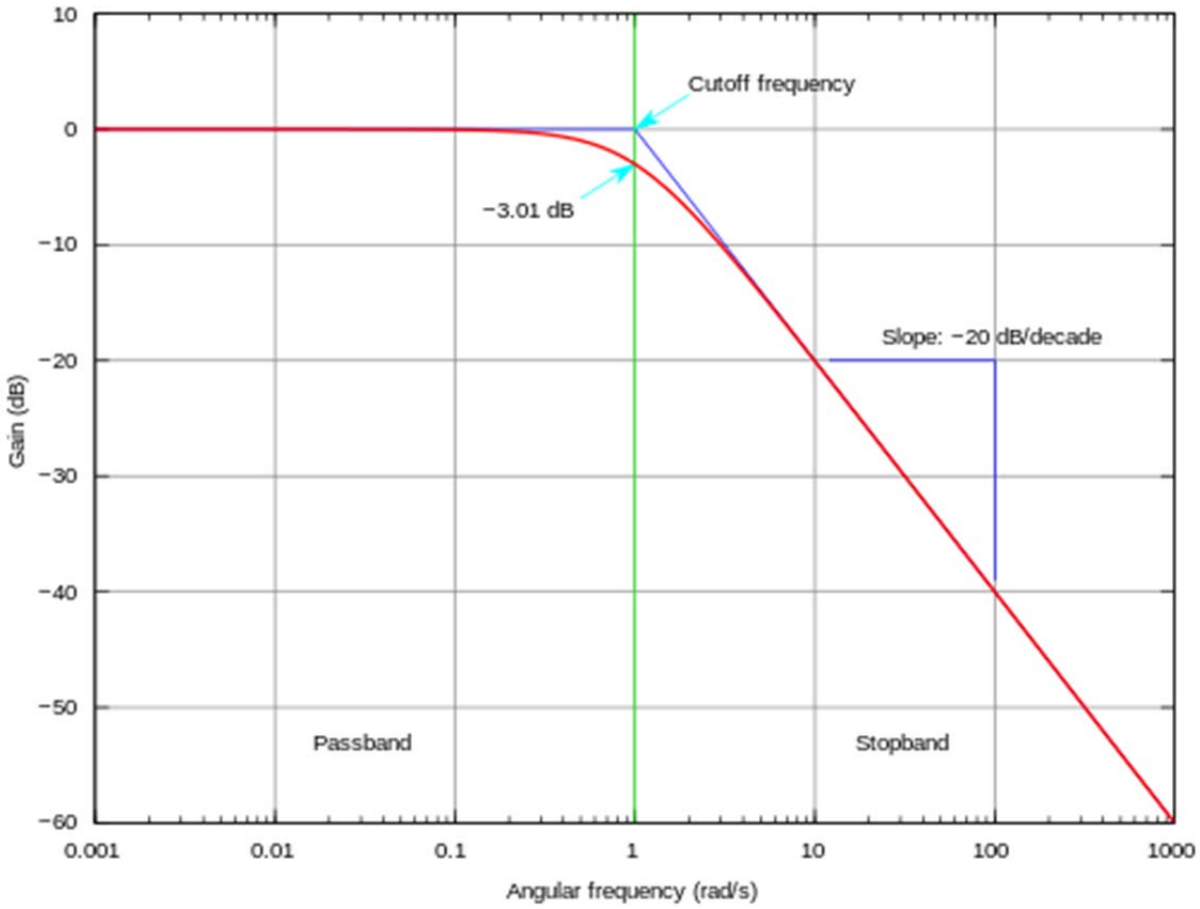


Figure 17. Generic 2-Pole Low-Pass Filter Characteristics

Typical full-scale output from Endevco Model 7270A piezoresistive accelerometers (10) is +/-100mV when using a 5V excitation, and so requires at least 25V/V gain to match the +/-2.5V input of the Analog-to-Digital Converter. Customers may use accelerometers that require higher gains, and for this the AE Recorder has a maximum practical 320V/V gain. Because the filter circuit is implemented as inverting, the accelerometer is connected inverted with the positive Wheatstone bridge output tied to the inverting input of the instrumentation amplifier. In addition to a passive RC filter ahead of the amplifier, the instrumentation amplifier also accommodates a wide source resistance from the sensor without affecting filter performance. Model 7270A accelerometers present a 650 ±300 Ohm load to the amplifier, but other types are in the 6k-Ohm range.

The Analog board's top-level hierarchical schematic is shown in Figure 18. The Microcontroller section parses I2C commands from the Digital board and sends SPI commands to the Programmable Gain devices and the Digital-to-Analog Converters. It also can shut down the analog circuit linear regulator to minimize power in a Delay Arm mode. The DACs generate the accelerometer balance voltages for each channel and the "fixed" 2.22V and 2.00V reference voltages needed to center the resulting analog signal around 2.5V for maximum ADC range.

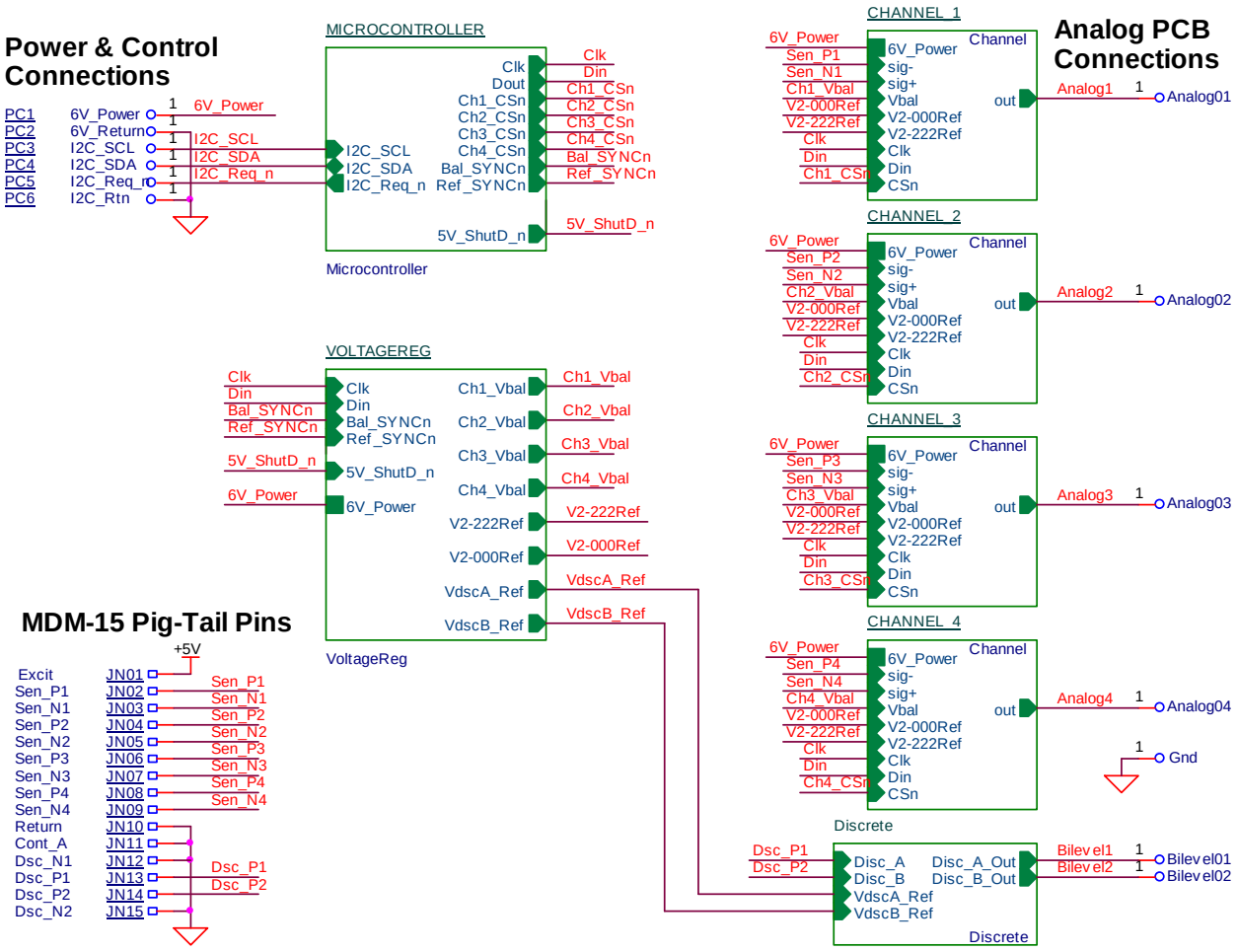


Figure 18. Top-Level 3A7000-002 Analog Board Schematic

Instrumentation Amplifier and Programmable Amplifier

Each channel includes a passive RC filter, instrumentation amplifier, programmable gain stage, and a filter section as shown in Figure 19. The circuit implements a 7-pole Phase-Compensated Butterworth low-pass filter with a 50-kHz DC bandwidth and a -3dB point of 57 kHz. The instrumentation amplifier's very high input impedance supports insertion of a single, shunt-capacitor pole preceding the first gain stage to attenuate high frequency signals that would otherwise introduce non-linearity due to slew rate limitations in the instrumentation amplifier (11; 12). The Analog Devices AD8224 instrumentation amplifier has a gain fixed at board assembly time by Resistor R2. For test Impact 1, the resistor was not installed so the first-stage gain was 1. For Crush 1, a 12.4k resistor is in place, setting the first-stage gain to about 5. The AD8224 common mode voltage maximum is the supply voltage minus 2.0V. To maximize the input range, the raw 6V supply powers the AD8224. Although the common mode range is not an issue for accelerometers, expanding the range increases flexibility in applying the AE Recorder to other sensors and inputs. The 49.9-Ohm resistor shown on the signal conditioning output in Figure 19 is combined with a 100pF capacitor located on the Digital board before the Texas Instruments ADS7265 Analog-to-Digital Converter. This 18 MHz pole is recommended in the ADS7265 data sheet to accommodate the minimum 80ns acquisition time Tacq for the device.

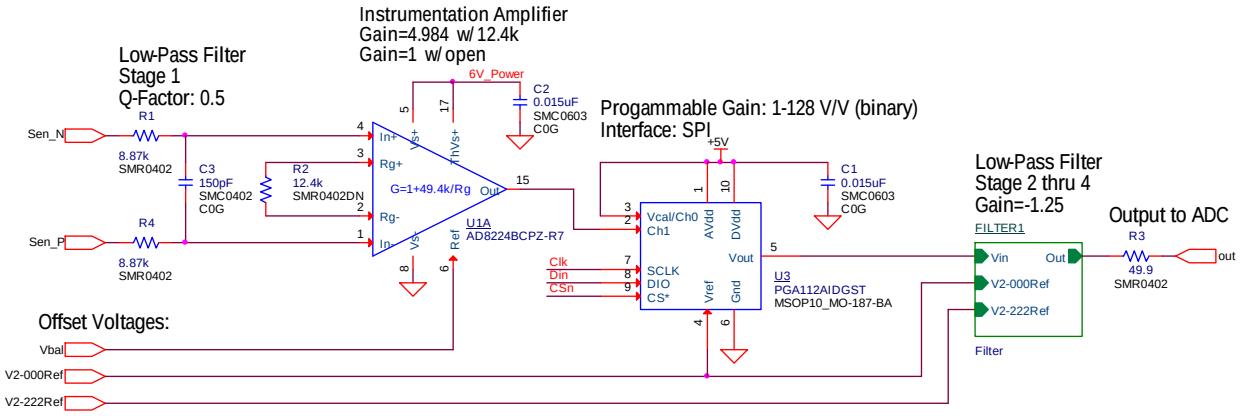


Figure 19. Signal Conditioning Per Channel

As input sensors change, the AE Recorder module must have adjustable gain to match. Also, flexibility is needed to change gain in the field during test article build-up. A programmable gain amplifier meets these requirements but is variable only in discrete increments. The Texas Instruments PGA112, shown in Figure 19, has gain in binary increments from 1 to 128. When combined with the rest of the fixed gain stages, gain from 6.25V/V to 800V/V is theoretically possible. Noise limits the practical gain to 400V/V or less. Control of the PGA112 is through the microprocessor's SPI bus.

Low-Pass, Anti-Aliasing Filter

A low-pass filter is necessary to avoid aliasing high-frequency content onto lower frequency, thus distorting the measurements collected. Switch-capacitor filters with sharp cut-off characteristics are available but have been found to display piezoelectric effects when used in high-shock instrumentation. The number of poles realizable using discrete amplifiers is limited, as is the available board area. A design compromise resulted with a 7-pole Butterworth filter that allows aliasing in the transition region between the Nyquist frequency and the sampling rate.

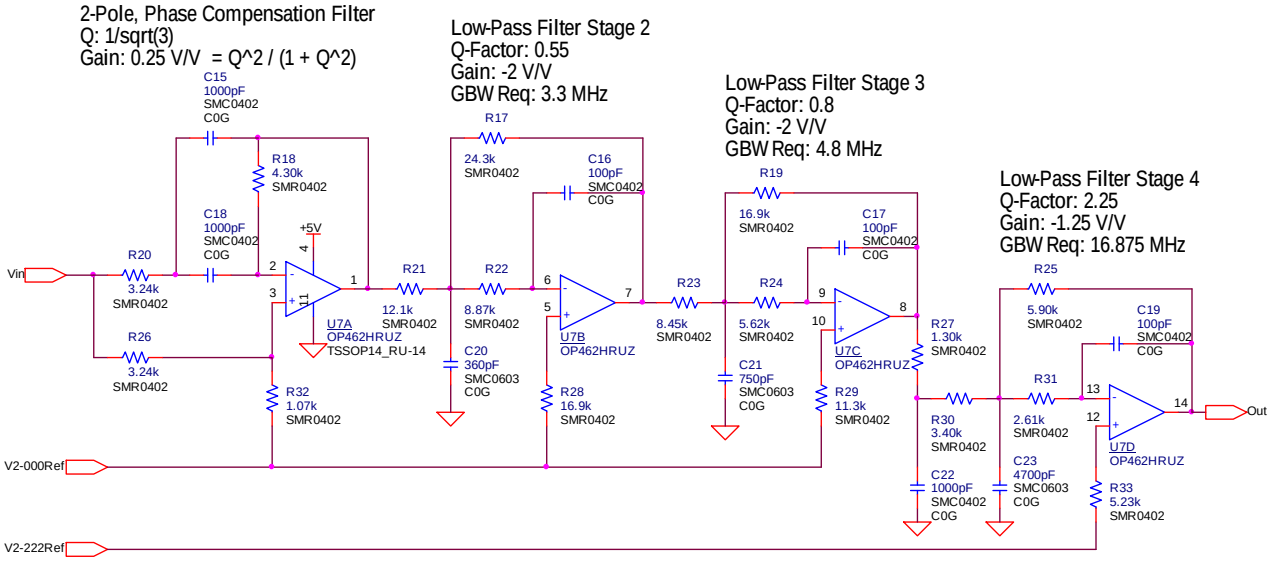


Figure 20. 57 KHz Phase-Compensated Butterworth Low-Pass Filter with -1.25 Gain

The phase compensated, Butterworth, MFB topology filter design was taken from the 3DDR-AM design (3) with few modifications and is shown in Figure 20. Note that the first pole is implemented as a passive RC stage preceding the instrumentation amplifier as discussed above. Analysis for the 3DDR-AM design ensured each amplifier had sufficient slew rate to support the gain-bandwidth product needed for its stage. Total power for the Analog board, excluding the microcontroller circuit and accelerometer excitation, is about 50mA at 6V. The design's 12-bit ADC requires 72dB attenuation at 200 kHz, which is the 250-kHz sampling rate minus the 50 kHz highest non-aliased frequency desired by the customer. We defined the pass-band as having less than 5% magnitude deviation and less than 5% phase nonlinearity. Thus, our 50-kHz pass-band was achieved with a 57-kHz -3dB corner frequency.

The AE Recorder has programmable gain to match accelerometer's wide sensitivity variation. Sensor balance must also be adjustable to compensate for accelerometer offset voltage. Two 16-bit Analog Devices AD5664 Digital-to-Analog Converters, shown in Figure 21, generate the offset voltage needed for each of the four input channels. In addition, 2.00V and 2.22V reference voltages were needed for all of the filter circuits, seen in Figure 20 as V2-000Ref and V2-222Ref.

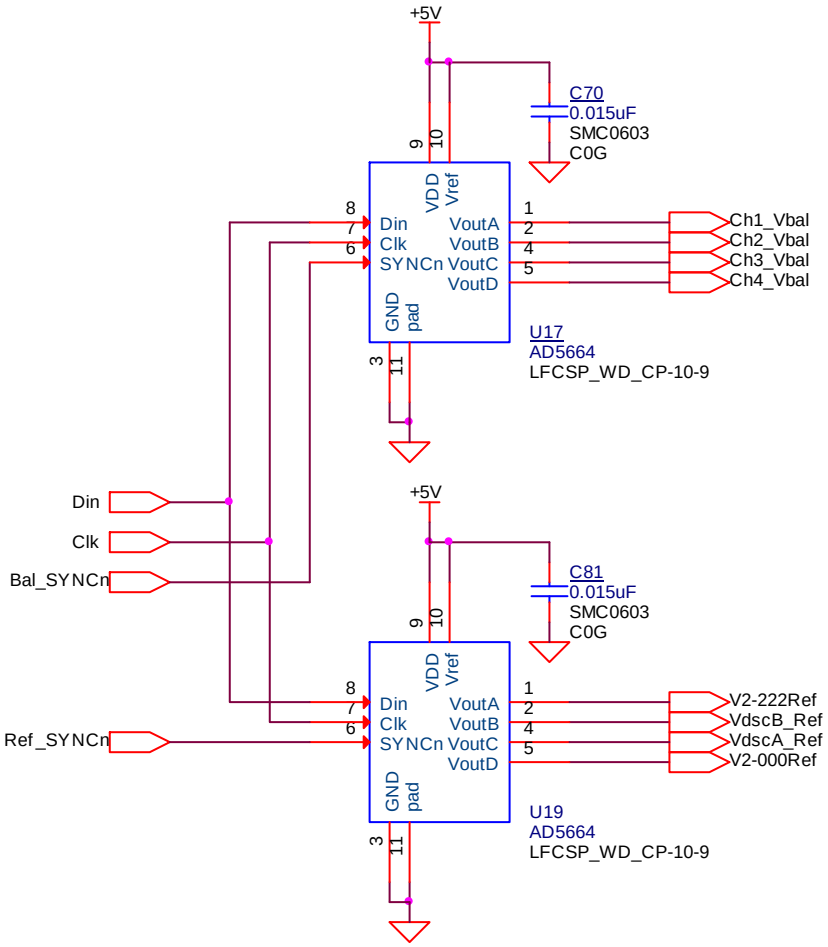


Figure 21. Digital-to-Analog Converters Adjust Balance and Threshold

Comparator Circuit for Discrete Signal Capture

Two other DAC voltages adjust the thresholds for the two discrete channels. The threshold should not exceed the 3.3V comparator supply voltage. This limit is enforced by the Digital board processor. The discrete signal input is protected with an input resistor and a Zener diode, as shown in Figure 22. The time constant of the RC filter is about 4 microseconds, selected to approximate the AE Recorder sample period.

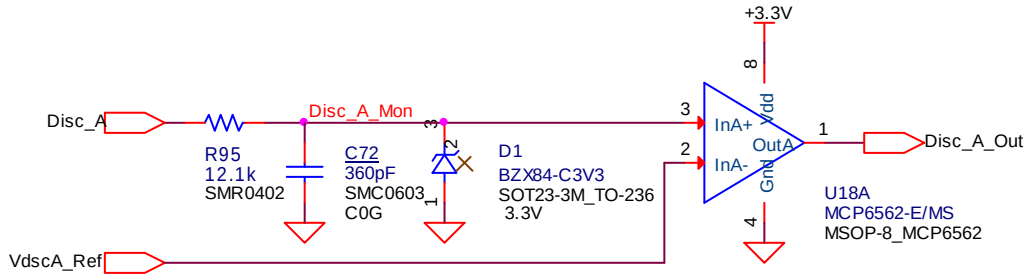


Figure 22. One of Two Discrete Channel Comparator Circuits

The discrete channel comparators are useful when events like contact closures are part of the data collected. An analog channel needlessly uses the limited memory capacity when simple fact-of-function data are all that is needed. As mentioned earlier, the threshold is programmable to provide additional flexibility. However, because the 0V to 3V range is relatively restrictive, external voltage dividers may be required to apply a signal within the comparator's range.

Microcontroller Configures Analog Board

The AE Recorder design included an I2C bus shared among all boards to configure the Analog and Power / Interface boards. A Microchip PIC24FJ64GB002 microcontroller (13) in a small, 28-pin QFN package accepts the I2C commands, qualifies them with the board address, and then controls the programmable gain and digital-to-analog converters using an SPI bus. A TXS0108 level translator converts the 3.3V microcontroller signal output to the 5V signal levels needed for the PGA112 programmable amplifier, as shown in Figure 23. Level translation is not needed for the AD5664 DAC, but is applied to use the same SPI data and clock signals for all. The 2-mm pitch, 5-pin programming connector has sufficient inter-pin spacing to be readily implemented for the test fixture. The 0.05" pitch programming connector typically used is too closely spaced.

A higher I2C pull-up resistance of 11.3k allows all four boards sharing the bus to be connected in parallel without over-loading the bus. The resistance was selected for convenience from among the values used elsewhere in the design to reduce the bill of materials. The service request signal I2C_Req_n was included in the event this feature was needed, and also uses a pull-up value from elsewhere in the design.

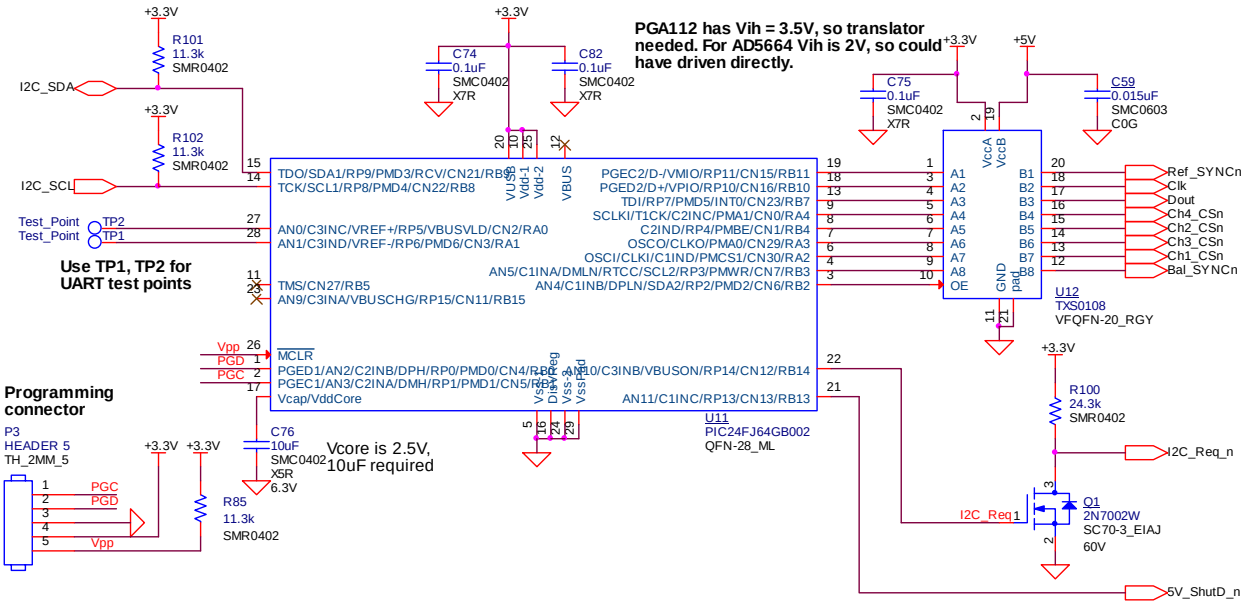


Figure 23. Analog Board Microprocessor Controls Gain and Balance

Op Codes sent from the Digital board using the I2C bus are listed in Table 1. The I2C data following the address (board number) are interpreted to configure the PGA and DAC, or to control the 5V analog power. The "Unlock Protected Parameters" command must precede the serial number and board number commands. None of the values, including serial number and board number, are written to the board are transferred to non-volatile memory until the "Write Volatile Parameters" command. Because the PGA and DAC are both powered from the 5V analog power, 5V power must have been enabled with the "Turn On 5V Power" command before any settings to the PGA or DAC are accepted.

Table 7. Analog Board Configuration Op Codes

Op Code	Bytes	Function
0x01	3	Unique serial number, 16-bit
0x02	2	Board number: 0, 1, or 2
>= 0x10 and <= 0x17	3	DAC routine, 16-bit value
>= 0x20 and <= 0x23	2	PGA routine, 8-bit value
0xA1	1	Turn on 5V power
0xAE	1	Turn off 5V power
0xC5	1	Read volatile parameters from nonvolatile memory
0xCA	1	Write volatile parameters to nonvolatile memory
0xD3	3	Unlock protected parameters, Op Code and 0x55, 0xAA

Sensor excitation voltage shares the 5V analog power on the board, and each Analog board has its own regulator. If a sensor draws too much current, the linear regulator will shut off which cuts 5V power to the entire Analog board and measurements from all four channels will be lost. Although 10V is still the default accelerometer calibration voltage, 5V is becoming more typical and is commonly used on battery-powered systems to reduce power consumption.

Control, Digitization, and Memory

The core measurement digitization and storage functions are contained in the 3A6998 Digital board. All the analog outputs from both the 3A6999 Power / Interface and 3A7000 Analog boards are connected here, and also the Analog board's discrete signal comparator outputs. An Actel Smart Fusion A2F500 Field-Programmable Gate Array (FPGA) orchestrates digitization and data movement through a short, internal First-In, First-Out (FIFO) buffer and then into non-volatile memory. The top-level diagram is shown in Figure 24, and the front and back of the circuit board shown in Figure 25. (The patch on the board resulted from an error on the ground plane Gerber file, and has been corrected in Version 003.)

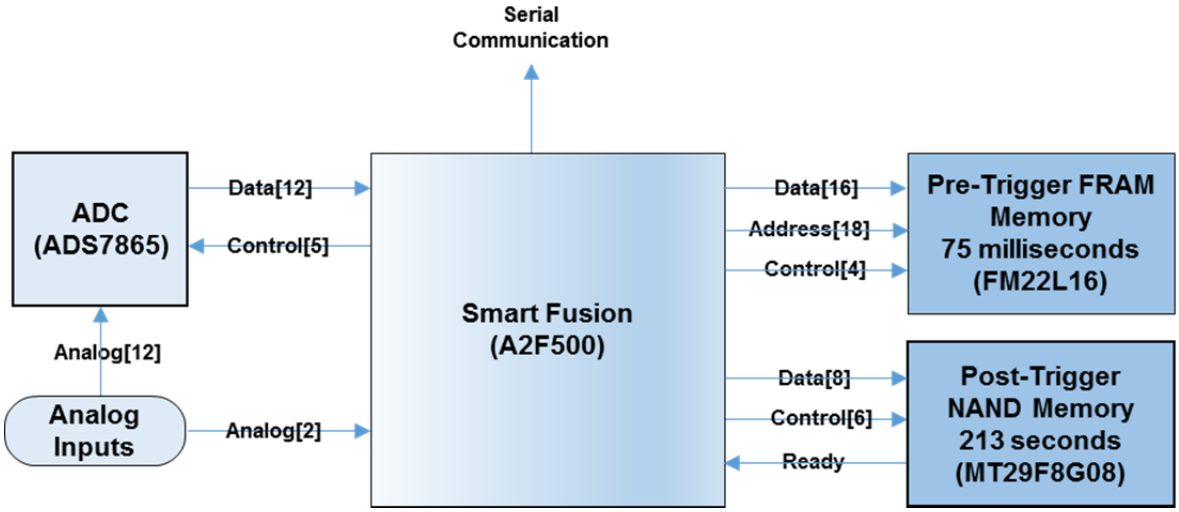


Figure 24 Block Diagram for Digital Control

The Actel Smart Fusion combined FPGA logic with ARM Cortex M3 core microcontroller reduces component count, board area, and design complexity. The microcontroller implements the necessary supervisory control which handles the user interface, recorder configuration, and the NAND Flash bad-block table. Aspects more efficiently handled with parallel logic include control logic to drive the memory addresses and ADC timing pulses, plus sequencing the internal FIFO buffering of ADC data acquired during NAND Flash programming time. The logic section also handles the high-speed data extraction serial interface.

During Arm mode while data are being acquired and stored, the microcontroller periodically monitors the quiescent value of all analog signals used to trigger the last phase of data capture. Because accelerometers in particular are strongly influenced by small temperature changes, the microcontroller updates the gradually-shifting quiescent reference levels used for the trigger comparison and stored in the Sampling Controller while the system waits for the abrupt change associated with the impact trigger. This patented automatic threshold (4) adjustment is particularly important during long periods with the unit armed and waiting for trigger.

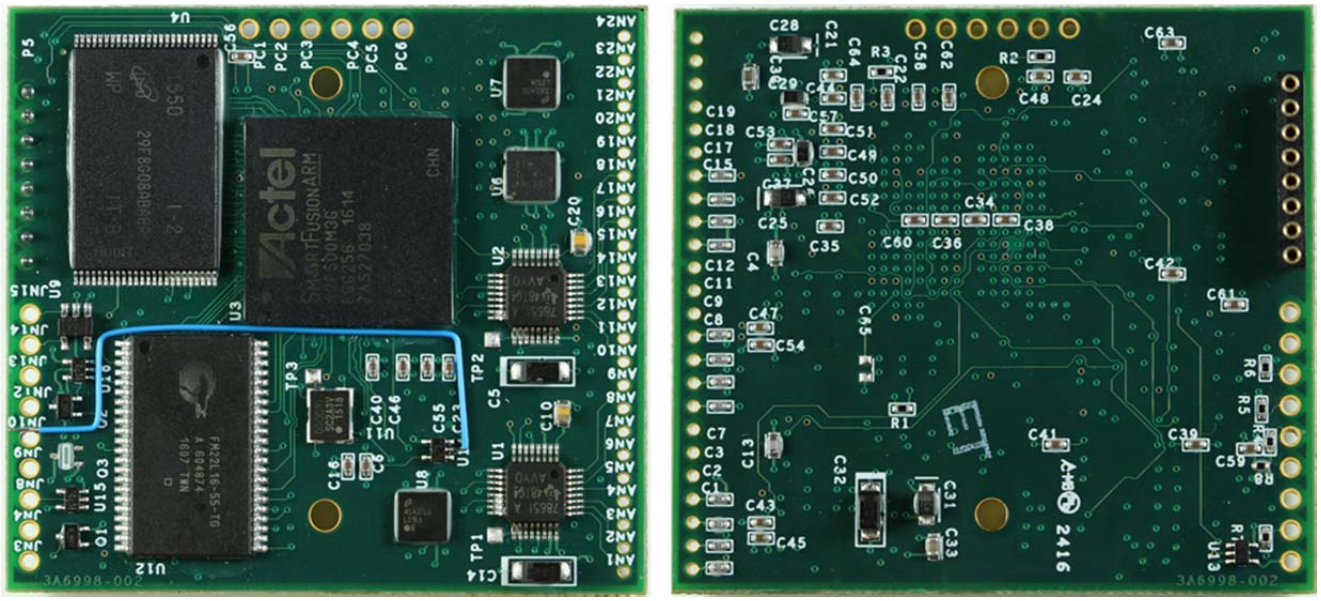


Figure 25. 3A6998-002 Digital Board

On power up, the AE Recorder enters the "User" low-power mode with all the 5V analog power and sensor excitation voltages off. A command over the external serial interface results in the microcontroller sending I2C messages to each Analog board that turns on the 5V power and initializes the gain (Programmable Gain Amplifier) and balance (Digital-to-Analog Converter) devices. The user interface software has scripts that send a sequence of commands to adjust balance on all the accelerometer channels.

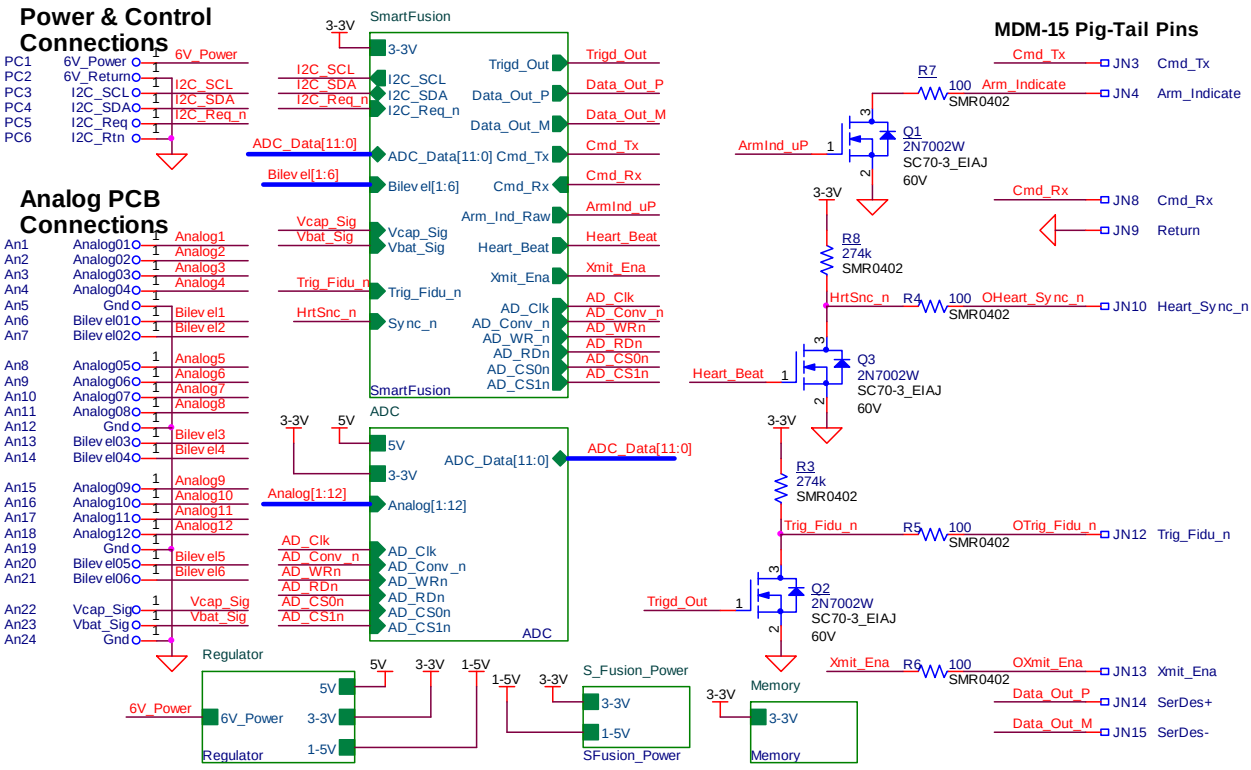


Figure 26. Digital Board Hierarchical Top Level

The 3A6998 Digital schematic top level, shown in Figure 26, indicates all the connections off the circuit board. The PC1 through PC6 connections in the upper left are the power and control bus common to all the AE Recorder boards, with the Digital board the I2C bus master. Although each of the other boards has an electrical connection for asserting the I2C_Req_n line, firmware was not implemented nor a reason for implementing it.

On the right side of the schematic are JN3 through JN15, which are signals in the MDM-15S harness split between the Digital board and the 3A6999 Power / Interface board. Connections on the Digital board focus on the serial interface (Cmd_Tx, Cmd_Rx, SerDes+, SerDes-) and special discrete signals (Arm_Indicate output, Heart_Sync_n input/output, Trig_Fidu_n input/output, and Xmit_Ena output).

The remaining through-hole connections An1 through An24 include the analog and discrete signals that mostly originate on the Analog board. Each of the three Analog boards produce four analog signals and two discrete signals. The total of twelve analog signals are routed to the ADC module shown in Figure 26. Dropping down the hierarchy into this block, the schematic section shown in Figure 27 contains the digitizer details. Each of the analog input signals has a 100pF filter capacitor, which when combined with the 49.9-Ohm resistor on the Analog board provides a passive 18-MHz low-pass filter. All of the capacitors on analog portion of the Digital board are either NPO or tantalum to minimize piezoelectric contamination of the analog signal. The Smart Fusion logic drives all the control signals to acquire each pair of channels in sequence.

The primary criteria for analog input signal routing to the ADC was minimizing routing layers and thus signal contamination. This meant the signals did not line up sequentially with channel digitization, and also did not line up with the gain and balance devices on the Analog board. These issues were compensated in the Smart Fusion microcontroller to map all of these to a consistent index which is listed later in Table 9 and Table 10.

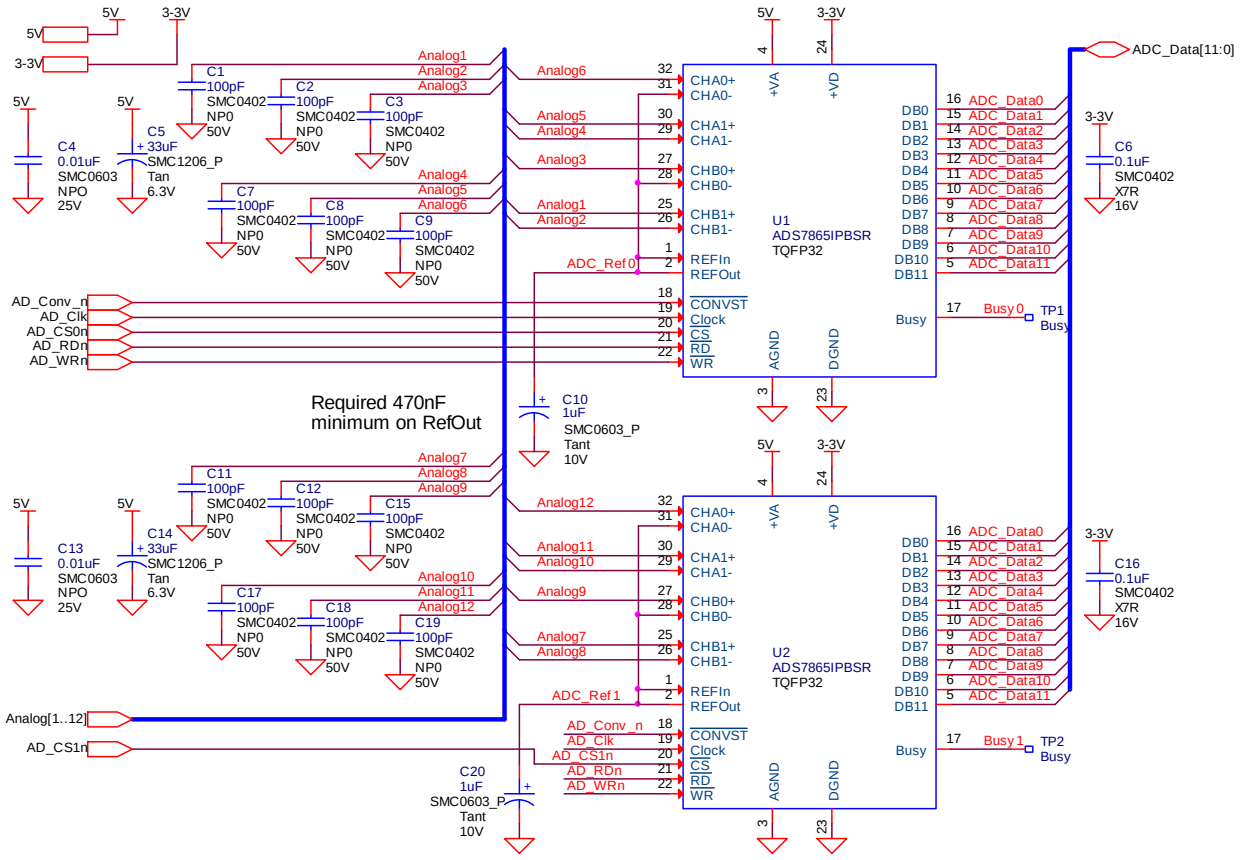


Figure 27. Analog-to-Digital Converter Schematic

The Memory block from the Figure 26 hierarchy view is shown as schematic details in Figure 28. The ADC_Data bus from the ADC output is routed through the Smart Fusion fabric into the NAND Flash and FRAM memory. Both memory types are needed to meet design requirements. FRAM has an advantage recording the pre-trigger data because it can be rewritten quickly during a possibly hours-long circular memory operation waiting for the brief pre-trigger recording, yet is non-volatile. However, capturing the 5-seconds required recording time requires a much larger memory: the NAND Flash. The small FIFO implemented in the FPGA is adequate to buffer continuous data collection during the programming periods of the NAND Flash. Thus, the FPGA can implement this function without additional external devices. Unfortunately, the FIFO has insufficient capacity to handle all the pre-trigger data so the FRAM is still needed.

The Micron NAND Flash MT29F8G08ABABAWP-IT with 1.074×10^9 Bytes (14) has a 500us worst-case write time for a 4096-byte block. This write time plus the transfer time to the Flash internal buffer limit the maximum AE Recorder sampling rate. The block contains 204 of the 20-byte sample sets. The total recording time is thus 213.96 seconds. The microcontroller maintains the Flash bad-block table and prepares the address for the next Flash block read during data extraction or written during data acquisition. The memory interface takes the best advantage of the FPGA by using parallel logic to efficiently handle data movement and internal FIFO buffering, while relying on the microcontroller function to select the next valid block.

Measurements must be continually captured at precise time intervals without interruption. The NAND Flash memory cannot accept a continuous stream of data during its addressing phase or 500-microsecond-long programming phase. All data destined for the NAND Flash pass through the FIFO before being transferred to the Flash internal programming buffer. The FIFO memory size is relatively small: only 4k bytes, equal to the size of one Flash write buffer. At 250k samples per second, the AE Recorder captures 20 bytes of data every 4 microseconds, and the FIFO holds 204 sample sets which corresponds to 816 microseconds.

Unless two NAND Flash chips are used in ping-pong fashion, the page programming speed limits the maximum sampling rate. Other Flash devices list a 500us ***typical*** page-programming interval, but have an unacceptably high maximum value and were therefore unable to keep up with data storage rate. The average programming time per byte cannot be less than the acquisition time per byte, which depends on the sample rate chosen. For a 250k samples per second rate, the Flash chip must be able to store at least 5M bytes per second. The MT29F8G08ABABA is able to store 4320 bytes in a maximum of about 675 us when using a 20 ns (50 MHz) clock, giving a satisfactory minimum input data rate of 6.4M bytes per second. (Because Flash memory can have data reliability issues, a Reed-Solomon encoding scheme creates error detection values stored in 86 of the extra 224 bytes beyond the normal 4096-byte page. The Reed-Solomon is also used to generate error detection codes when transmitting the data during data extraction. Both normal-speed and high-speed serial interfaces are included for control and data extraction, respectively.)

FRAM, Ferroelectric Random Access Memory, is named for the ferroelectric phenomena, although it is not actually affected by magnetic fields. FRAM is ideal for the pre-trigger buffer because it is non-volatile, random-access, can be rewritten trillions of times, and has no need to erase stored data before reusing its location. The Ramtron FM22L16 FRAM (15) is organized as 256k x 16 bit, which holds 26,214 of the 20-byte sample sets. At 250k samples per second, the capacity is 105 milliseconds, however, the design is configured to store only 75ms pre-trigger data, with the remaining capacity used to allow initial overlap between the FRAM-stored data and the Flash data.

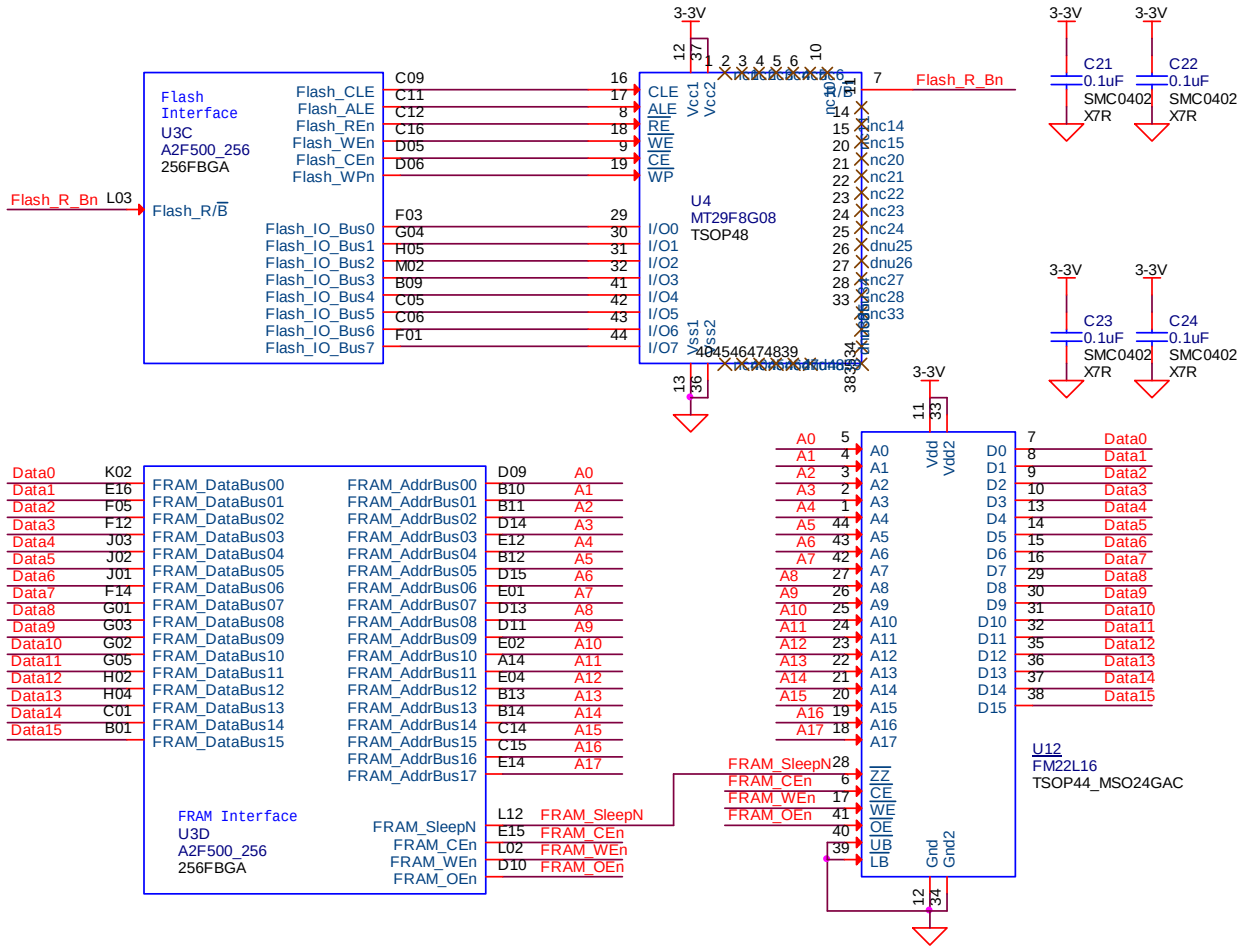


Figure 28. NAND Flash and FRAM Memory Interface

The schematic details of the SmartFusion hierarchical block are shown in Figure 29, which includes design blocks within the Smart Fusion FPGA and some external components such as the oscillator U11 and discrete buffers. The Smart Fusion FPGA uses an internal Phase-Locked Loop to generate 150 MHz using the Silicon Laboratories Si500S oscillator, programmed by the component distributor (Digi-Key) to 20 MHz.

As an example of an FPGA block, the I2C / serial interface module depicted near the oscillator allows the microcontroller in its supervisory control role to send configuration data to other system boards using the I2C interface. This module also contains the SerDes high-speed serial interface for rapid extraction, and the 115.2k Baud UART interface for external user control.

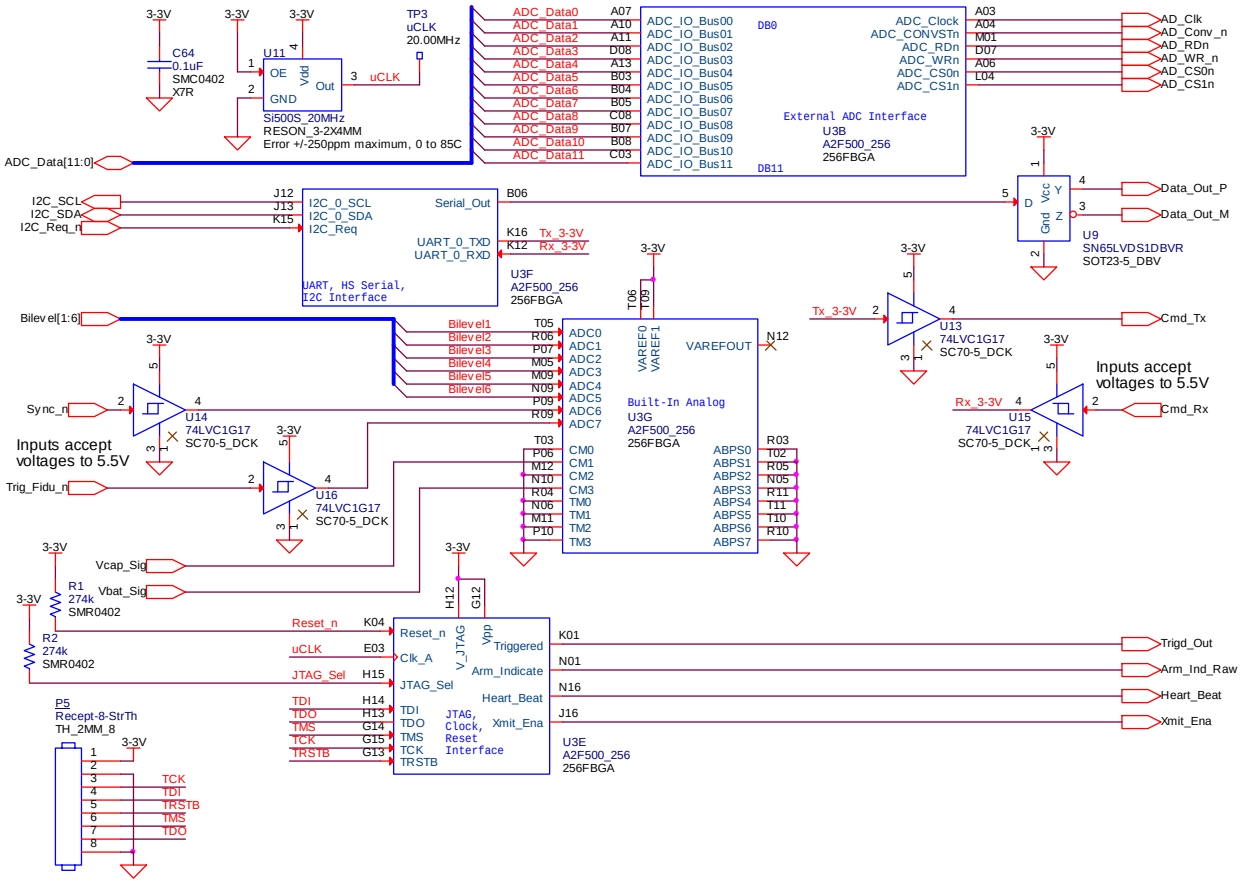


Figure 29. Smart Fusion Discrete Signal and Serial Interface

Communicating with Recorders

The recorder unit primarily communicates using a 115.2k Baud serial link with 8 bits, no parity, 1 stop bit, no flow control. Any of several interface boxes can be used to connect to it, but a QuickUSB interface is required to use SerDes data dump commands like *F*, *G*, and *g*. The commands are listed in Appendix A – Alphabetical Command List, Appendix B – Command List by Category, and Appendix C – Command Descriptions in Depth.

There are two potential operating modes for a recorder:

1. Controlled by a computer program, such as a graphical user interface (GUI)
2. Controlled by a human at a serial terminal

When a human is entering commands directly, it's easier if the person can see what they're typing. They can make sure each command is correctly entered, and use backspace to correct any mistakes. In this mode, the recorder echoes back every keypress it receives, as well as sending the responses from commands.

However, when a computer program is in control, echoed characters are unnecessary overhead. The program doesn't need to know what it's transmitted, only the response from the command. In this mode, the recorder only sends responses.

The recorder starts up in program-controlled mode, with input echoing off. To change modes, use command "?" to switch to human-controlled mode (and print a list of available commands), and command "/" to switch to program-controlled mode. It doesn't hurt anything to switch to the mode the unit is already in, and in fact it's expected that a user will use command "?" to get a list of commands while already in human-controlled mode.

Having a recorder in human-controlled mode when actually a program is in control will lead to strange results as the program tries to parse the echoed command as if it were a response, so always end with command "/" when you're done entering commands manually.

Although backspace works, the arrow keys don't. If a command must be edited, backspace is the only tool available. Also, there is no command history other than the screen buffer of your terminal program.

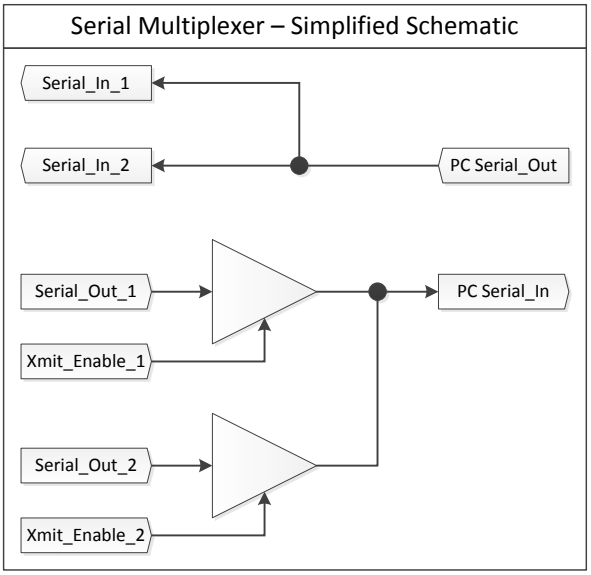
Issuing Commands

Each recorder unit needs to be issued an individual address, so that commands can be sent specifically to that recorder when it's in a multi-recorder system. An address is the first character sent following a newline (i.e. a press of the enter key, a transmission of '\n' from a program). A recorder will disregard any command not addressed to it.

For example, to send command *C* to a unit with address '1', type:
1C<enter>
Only the unit with address '1' will reply, with the response *K* (or *!*, if command *C* isn't allowed right now); if another unit with address '2' is listening on the same serial connection, it will ignore the command.

However, if you want to send command *C* to *all* units, use the special address '0':
0C<enter>
No units will reply (to prevent contention at the serial multiplexer), but all units will accept the command. However, depending on the units' states, the command might not have an effect for all of them; for *C*, it depends on whether or not they're armed. Be sure that all units are prepared to accept a particular command before you send it to all units, as you will receive no feedback if there's an error. Optionally, after issuing a command to all units, you can use individually-addressed diagnostic commands to see if the command was successful on each unit.

(In this context, for a unit to "reply" means to send a response back through a multi-recorder system containing a serial multiplexer. This involves the recorder not only transmitting a response on its serial output, but asserting its Xmit_Enable output to put that response on the output of the serial multiplexer. Every recorder will transmit a response on its serial output when it receives a command addressed to it (even with address '0'), but it won't assert its Xmit_Enable output unless specifically addressed – if multiple units asserted Xmit_Enable at once, there would be a conflict on the output of the serial multiplexer. One side effect of this two-signal system is that it's easier to debug a unit's behavior, as a unit's responses are always available from its serial port.)



Finally, it's possible to configure a unit to not have an address (see the in-depth description of command *+*). Such a unit will always act as if its address character has already been received. For example, to issue it the command *C*, simply send:

C

The unit will not reply to any command. Naturally this is only for single-unit debug and testing purposes, as a direct serial connection to the unit will be required to read the unit's responses; a serial multiplexer will never pass the messages along.

Argument Parsing Principles

- Field types are fixed. If a field requires a decimal value, you can't enter *0x10* if you want sixteen; enter *16*.
- Hexadecimal fields don't require any prefix or suffix. If the field description says that a hexadecimal value is required, just enter the hex digits, e.g. *c56* for 0xc56. Letter case doesn't matter to hex values, so *C56* works just as well.
- String fields are always at the end of a command, and may include any characters except newlines and backspaces. They are terminated by a newline (i.e. by reaching the end of the command) or by reaching the maximum length for the field (generally 32 characters).
- Fields that aren't string fields are terminated by a space character, by an invalid character (i.e. one that doesn't match the field type), or by reaching the maximum length, which is given in the field description.
    - Example 1: If a command requires a hexadecimal value, an argument of *0x12* will be parsed as a *zero* followed by an *invalid character*, and interpreted as "0", while *x12* is kept to be used as the next argument (if one exists).
    - Example 2: If a command requiring a decimal value is given an argument of *12ac*, it will be interpreted as "12", as "a" is not a valid decimal character; *ac* would be used for the next argument.
- Any number of spaces between arguments is valid, but tabs and other whitespace are not allowed. It's even legal to have zero spaces between arguments, but that's likely to have unexpected results unless you're very careful to make every argument the exact maximum length.
- Leading zeros are not necessary for decimal and hexadecimal arguments, but they are not ignored.
    - Example 3: If the command requires two argument fields of three bytes each, giving it *0 23* will be interpreted as "000000 000023" but *00000000 23* will be interpreted as "000000 000000" and the "23" will be dropped. The first 6 zeros will make up the three bytes of the first field (terminated by maximum length), the next two zeros will make up the second field (terminated by the space), and the command doesn't have any more fields for the "23" to go into.

- If extra arguments are given to a command, they are ignored. If not enough arguments are provided to a command, the response will indicate an error.
- If an argument is out of range, the response will indicate an error.

Command Responses

Each command has a response format given in the list of command descriptions. The response format shown in the list is the format of a successful response; in many cases, a successful response will be simply "*K*".

An unsuccessful (error) response will start and end with "*!*"; usually this takes the form of a simple single "*!*", but more complicated messages such as "*! 00000005 3 2066 !*" are possible for a few commands. (These messages explain exactly where in the firmware the error occurred, and are generally useful only to the developers.)

All responses are a single line, i.e. zero or more characters followed by a carriage return and linefeed (in C syntax, "\r\n"). (Empty lines are possible only for command *i*; other commands will always return at least one character before the newline.)

Hexadecimal fields in a response can have leading zeros, but decimal response fields won't.

User-defined strings, i.e. the arm string and unit description string, have a fixed length of 32. If a string provided by the user is less than 32 bytes long, the string will be padded when it appears in a response. The pad bytes will be null characters (bytes of value 0x00), which makes printing such responses slightly tricky, as using printf("%s") will end the printing when the nulls start; fwrite() is one way to make sure that all the characters get printed.

The debug menu command, command *J*, contains exceptions to most of these rules.

Retrieving Data

Recorded data can be downloaded from the recorder either via low-speed serial (the same link used for commands) or high-speed SerDes. Making a SerDes connection requires a particular type of interface box, containing a QuickUSB chip.

SerDes is about a thousand times faster than low-speed serial when data is being moved, but this doesn't include the time spent issuing commands or decoding and storing the received data. To increase dump speed, reduce overhead by requesting many pages at once over SerDes, for example 256, 512, or 1024 pages per command.

Operating Modes

The recorder has several operating modes, which affect (among other things) the commands the unit will accept. All modes except **Recording** can be cancelled with command *f*, which returns the unit to **User** or **Locked** mode. The current mode is given by commands *s* and *W*.

Table 8. AE Recorder Operating Modes

Mode (cmd *s*)	Submode (cmd *W*)	Description
USER	U	**User** – The unit isn't doing anything. This is the default mode.
	L	**Locked** – The unit's memory contains recorded data; to protect it, all commands that change the contents of memory are locked out (i.e. will return an error) until command *E* is issued.
	D	**Delay-Arm** – The unit is in a low-power state with signal amplifiers and accelerometers off, counting down a timer until it switches to **Warmup** mode. Command *d* gives the remaining time. Command *S* will return bad values for analog inputs and bilevel inputs in this state.
ARM_FIRST	W	**Warmup** – The accelerometers have been turned on and are warming up. Trigger events are ignored. Command *l* can set how long this lasts. Command *d* gives the remaining time until the unit switches to **Arm-First** mode.
	F	**Arm-First** – The unit is now truly armed and is storing pre-trigger data to FRAM. Any trigger event will begin **Recording**.
RECORDING	R	**Recording** – The unit has been triggered. Data is being stored to flash. How much data will be stored is set by command *n*. When recording is complete, the next mode depends on whether multi-trigger is enabled (command *t*); the next mode will be **Arm-Again** if multi-trigger is enabled, **Power-down** otherwise. If flash memory becomes completely filled, the unit will go to **Power-down** mode immediately. This is the only mode that cannot be force-exited with command *f*; only command *R* can stop this mode before it's finished.
ARM_AGAIN	A	**Arm-Again** – The unit is waiting for another trigger. When a trigger event occurs, the unit will enter **Recording** mode. Unlike **Arm-First**, no data is stored to FRAM while the unit is in this state.
POWER-DOWN	P	**Power-down** – The unit is finished recording. When the unit enters this state, it can optionally turn itself off; this is controlled by command *;*. If the unit does not turn itself off, it will not stay in this mode but will enter **User** mode.
Others	O	Other modes exist, but they are for short operations performed by some commands, which end before another command can be run. The only modes that should be seen by the user are listed above.

Recorder Configuration

The recorder contains two sets of configurations: one in RAM, and one in nonvolatile memory.

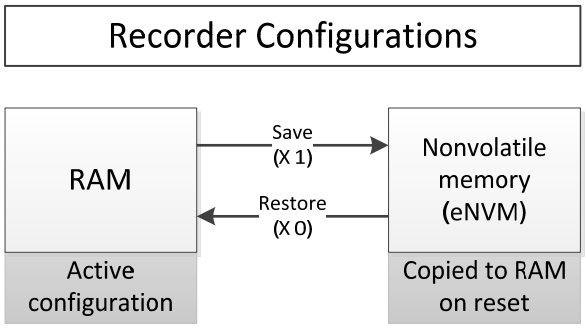


Figure 30. Recorder Configurations Diagram

The configuration in RAM is the current, active one, controlling what the recorder does; it is lost when the recorder is reset or turned off. The configuration in nonvolatile memory is a backup; it is copied to RAM when the recorder is reset or turned on.

Command *X* is used to move configuration data between RAM and nonvolatile memory. With a nonzero argument, it will save the current configuration. With an argument of zero, it will restore the saved configuration, overwriting the current configuration.

Resetting the recorder with command *R* will also reload most settings from nonvolatile memory. The exceptions are the settings stored on analog boards, controlled by commands *V*, *Y*, *y*, and *Z*.

Many commands change part of the recorder's configuration. Some of these commands save their changes to nonvolatile memory immediately, but other commands only store their changes in RAM. The difference is that some commands' settings might be changed a lot before the user decides on a good value; these commands are designed to only affect RAM. Once a good value is found, it can be saved to nonvolatile memory. Other commands, meanwhile, aren't used often and aren't useful for experimentation; these commands save their changes immediately.

Recording Data, Arming, and Triggering

At the very least, the following must be done for a successful data-recording session:

1. Unlock write access to the flash: *E*
2. Clear the existing data from the flash: *b*
3. Optional: Set the length of the recording period: *n <pagesPerTrigger>*
4. Arm the unit: *A [armString]* or *D <delaySeconds> [armString]*
5. Recording will begin when the unit is triggered, either manually via command *C* or due to a trigger condition being satisfied by an input

When an arming command is sent to a recorder, there are several periods of time that must pass before it is ready to be triggered and store data:

1. Delay-Arm (only if arming happens using arm-on-power-up or command *D*)
2. Warmup (length is controlled with command *I*)

Only when the Warmup period is complete will the unit respond to triggers, either input-based or caused manually with command *C*.

Triggering on a given input channel requires:

- The unit is in an armed state (Arm-First or Arm-Again) (check with *W*)
- The input channel is enabled as a trigger source (set with *M*)
- At least one of the channel's two thresholds is enabled (set with *m*)
- An enabled threshold must be satisfied for the minimum number of consecutive records (set with *q*)

A threshold being satisfied means, for an analog channel, that the measured value is less than the low threshold (quiescent value – offset) or greater than the high threshold (quiescent value + offset). Threshold offset is set with command *o*. (A measured value equal to the threshold does not satisfy it, so a sufficiently large offset makes a threshold impossible to satisfy.) For a digital channel, the low threshold is satisfied by a low value and the high threshold is satisfied by a high value; the cutoff voltage between "low" and "high" values is set with command *Z*.

Channel Indexing

Most commands that take a channel as an argument require a channel index. The inputs are assigned to channel indices in the following way. (The ordering of the analog input indices is due to the order in which the channel inputs are wired to the ADC chips.)

Table 9. Analog Channel Indexing

Channel Type	Input	Channel index
Analog	Analog1	5
	Analog2	4
	Analog3	3
	Analog4	1
	Analog5	2
	Analog6	0
	Analog7	11
	Analog8	10
	Analog9	9
	Analog10	7
	Analog11	8
	Analog12	6
Digital	Bilevel1	12
	Bilevel2	13
	Bilevel3	14
	Bilevel4	15
	Bilevel5	16
	Bilevel6	17
Fiducial digital	Fiducial7 (Heart_Sync_n)	18
	Fiducial8 (Trig_Fidu_n)	19
Manual trigger	*C* command	20

For convenience, a table of just the analog inputs from the above table, sorted by channel index:

Table 10. Analog Input Sorted by Channel Index

Channel index	0	1	2	3	4	5	6	7	8	9	10	11
Analog input	6	4	5	3	2	1	12	10	11	9	8	7

Time Reporting

The time base of the recorder when storing data is based on the rate at which records are taken. Each record requires a minimum of 167 processor cycles to acquire and store; an additional number of cycles can be added to that to produce a variable recording rate. The clock rate of the main processor is 50 MHz, so the default number of additional cycles is 33, to store 250,000 samples per second; at 0 additional cycles, the recorder can theoretically store about 299,400 samples per second, but this mode has not been rigorously tested. The number of additional cycles can be read and written with command ***Q***.

Time is measured by the recorder in units of records, to which an FRAM address can be directly translated; since the FRAM is filled in about 0.1 seconds, a larger scale timer is also needed. Much like the hour and minute hands on a clock, the wraparound counter (aka "loops" counter) increments each time the FRAM write location reaches the end of FRAM and wraps around.

The duration since arm for a given FRAM address and wraparound count is:

```
(floor((FRAM address [17..3]*8+7)/10)
 + 26214 * FRAM wraparound count [19..0]) records
  * 20 ns/cycle * (167 cycles/record + betweenRecordDelay)
```

Assuming the default ***betweenRecordDelay*** of 0x21, one record is sampled every 4000 ns.

Only some of the bits of the FRAM address are used in the equation above; this is because only those bits are stored when a trigger happens. To use the equation with the output of command ***e***, use the ***position*** value the command returns in place of `(FRAM address [17..3]*8+7)` and ***loops*** in place of `FRAM wraparound count [19..0]`.

Firmware Version Notes

Firmware version 2014 and below

In these versions, command ***n*** only returns a single field, which is ***pagesPerTrigger***. The ***secondsPerTrigger*** field is not present.

Firmware version 2013 and below

In these versions, commands ***Y*** and ***y*** have a different type of channel selection field. For these commands only, the first command field, ***channel***, is not a channel index, but is instead the externally-visible analog input number, minus one. That is, to refer to input AnalogN, the value of ***channel*** should be N–1. For example, for a ***channel*** of 4, the selected input would be Analog5. Also, the output of command ***s*** is much less detailed, containing fewer subfields. Finally, flash-accessing commands such as **_** and ***3*** can be invoked outside of USER mode, which will cause the unit to change modes; this is a problem if the unit is armed at the time.

Firmware version 2012 and below

These versions lack the **~** command. The hardware they were designed for only had one fiducial signal, Trig_Fidu_n, which was connected to both fiducial channels; the heartbeat signal was not

recorded. The two fiducial channels were only linked in terms of their input, so changing the trigger configuration of one fiducial would not affect the trigger configuration of the other.

Firmware version 2011
This version has an *S* command with a different output order – the analog values are sorted by external channel number (from analog1 to analog12), rather than by internal channel index (from analog6 to analog7, see the Channel Indexing section). For backwards compatibility, issuing *S* with a nonzero argument will sort the returned channel values in the same order as all other firmware versions.

Also, the analog configuration values saved in eNVM are sorted by external channel number; this causes saved configuration values to be used for different channels if changing versions to/from version 2011, since the raw data in eNVM is not modified by version changes. This also means that commands *M*, *m*, *o*, and *q* use the external channel number to determine analog channel bit positions and channel selection.

Avoid using this firmware version if possible.

Firmware version 2008
This version has a bug in its SERDES flash dump function. If there are any bad blocks, the wrong parts of flash will be skipped during dumping, and there will be some duplication of pages. The low-speed serial flash dump command *3* works correctly, though it is rather slow; using it to dump flash overnight or over the weekend may be preferable to manually undoing the effects of the bug.

Avoid using this firmware version if possible.

Data Structure Definitions
The subsequent discussion of the FIFO and transfer of data into the Flash is clarified with a few definitions of various structures used in this design and their sizes.

Record
20-byte structure. Usually contains one sample of all analog/digital input channels plus some status flags, but may instead contain housekeeping data if multiple-trigger mode is enabled. See the Data Records section for details.

Glob
12 records. Size is 240 bytes, the most records that fit into a single Reed-Solomon encoding operation. (After encoding, the glob is 246 bytes long including the ECC bytes; if you're reading from flash or via SERDES, you'll get 246-byte globs.)

Table 11. NAND Flash Memory Glob Definition

Glob bytes	Contents
0-19	Record 0
20-39	Record 1
40-59	Record 2
60-79	Record 3
80-99	Record 4
100-119	Record 5
120-139	Record 6
140-159	Record 7
160-179	Record 8
180-199	Record 9
200-219	Record 10
220-239	Record 11
240-245 (optional)	Reed-Solomon ECC (flash and/or SERDES only)

Chunk
204 records, i.e. 17 globs. This is the most records that fit into a single page, either FRAM (raw, 4080 bytes) or flash (after Reed-Solomon encoding, 4182 bytes). The SRAM holds one chunk.

Page
4096 bytes in FRAM, 4182 bytes in flash. (A flash page can hold up to 4320 bytes, but only the first 4182 are used, as there's not enough room for another glob.) The native about-4-kilobytes unit for a given memory system, a page holds at least one chunk.

Pages are padded to a multiple of 256 bytes when sent over SERDES, for USB-related reasons. FRAM pages are Reed-Solomon encoded before SERDES transmission; this involves padding to the next glob boundary before being encoded, then padding to the next 256-byte boundary before SERDES transmission.

Block
128 pages, about 512K. The FRAM holds one block.

Trigger housekeeping data
8 bytes of non-record data describing the circumstances of a trigger. They store the cause of the trigger and the time after initial arming that the trigger occurred. The first trigger's data is stored in FRAM; subsequent triggers (if enabled) have their housekeeping data padded to the size of a record for storage in flash.

Understanding Recorded Data

Table 12. Data Record Structure Organized in 1-Byte Increments

byte	contents
0	analog input 6 [7..0]
1	analog input 12 [3..0], analog input 6 [11..8]
2	analog input 12 [11..4]
3	analog input 3 [7..0]
4	analog input 9 [3..0], analog input 3 [11..8]
5	analog input 9 [11..4]
6	analog input 4 [7..0]
7	analog input 10 [3..0], analog input 4 [11..8]
8	analog input 10 [11..4]
9	analog input 2 [7..0]
10	analog input 8 [3..0], analog input 2 [11..8]
11	analog input 8 [11..4]
12	analog input 5 [7..0]
13	analog input 11 [3..0], analog input 5 [11..8]
14	analog input 11 [11..4]
15	analog input 1 [7..0]
16	analog input 7 [3..0], analog input 1 [11..8]
17	analog input 7 [11..4]
18	status bits [7..0]
19	digital inputs [7..0]

Table 13. Data Record Structure Organized in 3-Byte Sequence

byte offset	byte 2 [7..0]	byte 1 [7..0]	byte 0 [7..0]
0	analog input 12 [11..0]		analog input 6 [11..0]
3	analog input 9 [11..0]		analog input 3 [11..0]
6	analog input 10 [11..0]		analog input 4 [11..0]
9	analog input 8 [11..0]		analog input 2 [11..0]
12	analog input 11 [11..0]		analog input 5 [11..0]
15	analog input 7 [11..0]		analog input 1 [11..0]
18	---	digital inputs [7..0]	status bits [7..0]

(The internal order of the analog inputs is different from the input numbers, due to the order in which the channel inputs are wired to the ADC chips. This numbering is seen in the response from the 'S' command, which orders the analog channels by internal index. The correspondence is as follows.)

Sorted by analog input:

Analog input	1	2	3	4	5	6	7	8	9	10	11	12
Internal index	5	4	3	1	2	0	11	10	9	7	8	6

Sorted by internal index:

Internal index	0	1	2	3	4	5	6	7	8	9	10	11
Analog input	6	4	5	3	2	1	12	10	11	9	8	7

Digital inputs definition
Bit 7: The "trigger detected" fiducial. If all recorders in a system have their OTrig_Fidu_n pins connected to a common line, this fiducial is pulled low when any recorder is triggered. Even the recorder that was triggered sees this low-going pulse, which lasts for just over 4 sample periods. A lone recorder will see 4 consecutive zero values on this channel (starting immediately after the triggering record), but in a multi-recorder system the interactions between recorders could cause 5 or more consecutive zero values to be present.
Bit 6: The "synchronization" fiducial. Pulled low by the Heart_Sync_n pin, which itself can be optionally driven by a 1Hz heartbeat signal. In a multi-recorder system, connecting all the recorders' Heart_Sync_n pins to a common signal source (perhaps external) provides a data alignment fiducial unrelated to trigger events.
Bits 5..0: Digital input channel values. Bit 0 is digital channel 1, etc.

Status bits definition
Bits 7..2: One fourth of the internal ADC data. Each group of four consecutive records contains the full 24 bits of internal ADC data, 12 bits from each ADC.
Bit 1: PLL locked state (1=locked). If it's unlocked, the system clock isn't running at 50 MHz anymore, and the timing and integrity of the record's data is suspect. This *should* never happen.
Bit 0: Different meanings in FRAM and flash. In FRAM, it's the FRAM seam detect bit (differing values in consecutive records indicates where the FRAM stopped recording); in flash, it's an index bit for the internal ADC data (1 in the first of the group of four records, 0 in the other three records).

Internal ADC data – part of the status bits
The internal ADCs (built into the SmartFusion), which measure battery and capattery voltages, are slower than the external ADC chips, which measure the analog input channels. The internal ADCs take data once every four recording periods. That data is then spread out over four consecutive records. Which of the internal ADC data bits are stored in a given record is determined by something we call a "record index", which represents how many 6-bit half-values have already been stored in the current 4-record group. The record index for a given record is not stored anywhere; it must be determined from the contents of nearby memory (in flash) or the record's contents and address (in FRAM).

Internal ADC storage in flash
In flash, there's no guaranteed connection between memory address and record index because there might be housekeeping records present at lower addresses, which store a record's worth of data to flash without affecting the record index. This makes a separate indicator necessary, namely status bit 0. In flash, status bit 0 is high only when the record index is 0, i.e. when a new set of 4 records is beginning. This lets a reader start anywhere in flash and be able to start reporting accurate internal ADC data within 7 records, by simply ignoring the internal ADC data bits until a high status bit 0 is found and the record index is known. (Note that housekeeping records also have the bit equivalent to status bit 0 set high, as part of their padding!)

Internal ADC data storage across consecutive records in flash:

Status bit 0 (in flash)	Record index	Status bits 7..2
1	0	Internal ADC 0 (Vcap) [11..6]
0	1	Internal ADC 0 (Vcap) [5..0]
0	2	Internal ADC 1 (Vbat) [11..6]
0	3	Internal ADC 1 (Vbat) [5..0]

Internal ADC storage in FRAM

In FRAM, the correspondence of the contents of status bits 7..2 to record index is the same, but status bit 0 has a different meaning. Status bit 0 is generally constant from record to record, changing its value only when recorded data reaches the end of FRAM and wraps around. Since the number of records that fit in FRAM, 26214, is divisible by 2 but not by 4, record index in FRAM can be directly determined from a record's address and status bit 0. Specifically, record index can be obtained by combining two bits of a record's first FRAM address with the value of status bit 0 using this algorithm:

record_index = FRAM_address[2..1] XOR (status_bit_0 << 1)

Table 14. Internal ADC Data Storage in FRAM across Consecutive Records

FRAM address	Status bit 0 (FRAM)	Record index	Status bits 7..2
0bxx…xx000	0	0	Internal ADC 0 (Vcap) [11..6]
0bxx…xx010	0	1	Internal ADC 0 (Vcap) [5..0]
0bxx…xx100	0	2	Internal ADC 1 (Vbat) [11..6]
0bxx…xx110	0	3	Internal ADC 1 (Vbat) [5..0]
	…	…	…
0bxx…xx100	1	0	Internal ADC 0 (Vcap) [11..6]
0bxx…xx110	1	1	Internal ADC 0 (Vcap) [5..0]
0bxx…xx000	1	2	Internal ADC 1 (Vbat) [11..6]
0bxx…xx010	1	3	Internal ADC 1 (Vbat) [5..0]

Housekeeping Data (Trigger Descriptions)

The trigger source and trigger time are stored as trigger housekeeping data in FRAM for the first trigger and in flash for all subsequent triggers.

Table 15. FRAM Addresses for Trigger and Housekeeping Data

address	word contents
0x3fffc	0, FRAM address [17..3]
0x3fffd	FRAM wraparound count [15..0]
0x3fffe	Trigger origin [11..0], FRAM wraparound count [19..16]
0x3ffff	0000000, Trigger origin [20..12]

Table 16. NAND Flash Pseudo-Record Structure for Trigger and Housekeeping Data

byte	contents
0	0xfc
1	0x96
2	0x30
3	0x03
4	0x69
5	0xcf
6	FRAM address [10..3]
7	0, FRAM address [17..11]
8	FRAM wraparound count [7..0]
9	FRAM wraparound count [15..8]
10	Trigger origin [3..0], FRAM wraparound count [19..16]
11	Trigger origin [11..4]
12	Trigger origin [19..12]
13	0000000, Trigger origin [20]
14	0xfc
15	0x96
16	0x30
17	0x03
18	0x69
19	0xcf

Bytes 0-5 and 14-19 are constant padding, intended to be a distinctive marker of a housekeeping record. Note that the equivalent of the status byte, byte 19, has bit 0 set; this bit is therefore not an infallible indicator of record index 0 in flash. The possibility of a housekeeping record must also be tested by seeing if the mentioned bytes possess the constant values defined above.

Trigger origin flag bits

The flag bits in the trigger origin bit vector correspond exactly to channel indices; see the Channel Indexing section for more information, but the short version is:

Bits 0-11: analog channels index 0-11 (internal channel indexing, not external channel numbers)
Bits 12-17: bilevel (digital) channels 1-6
Bit 18: Heart_Sync_n fiducial digital channel
Bit 19: Trig_Fidu_n fiducial digital channel
Bit 20: force-trigger from microcontroller

If a bit is set, then the corresponding source caused the trigger event. It's possible for multiple bits to be set, indicating that multiple sources caused a trigger event simultaneously, but this requires multiple trigger conditions to be satisfied by a single record.

Timekeeping Principles

Time elapsed since arming is derived from the FRAM address and FRAM wraparound count in the following way:

Since the FRAM address increases by 10 for each record that is stored (a 20-byte record fits in 10 FRAM words), the lowest 3 bits of the FRAM address are unnecessary to uniquely determine each record's address. (The unknown three bits define an interval of 8 addresses. Only one boundary between groups of 10 addresses can fit into that interval, maximum.) For a given FRAM address A=0bxxxxxxxxxxxxxxx000, the number of records that begin prior to that address is floor((A+7)/10). Because the time taken to store each record is defined by the software (by adjusting the clock cycles of stall between sampling cycles), the FRAM address can be converted into the time elapsed since the most recent write to FRAM address zero, serving as a short-term clock. For this reason, the FRAM address counter is kept running, even when no data is being stored to FRAM anymore.

Similarly, the FRAM wraparound count serves as a long-term clock. Given the number of records that fit in the FRAM, the time taken for each wraparound can be easily derived, and the number of wraparounds multiplied by that time to give the larger-scale offset from the very first FRAM write. The number of records that fit in FRAM is floor((FRAM_BYTE_CAPACITY-8)/20), or 26214 when using the Ramtron 22L16. Each wraparound thus takes time equal to (user-defined record duration)*26214.

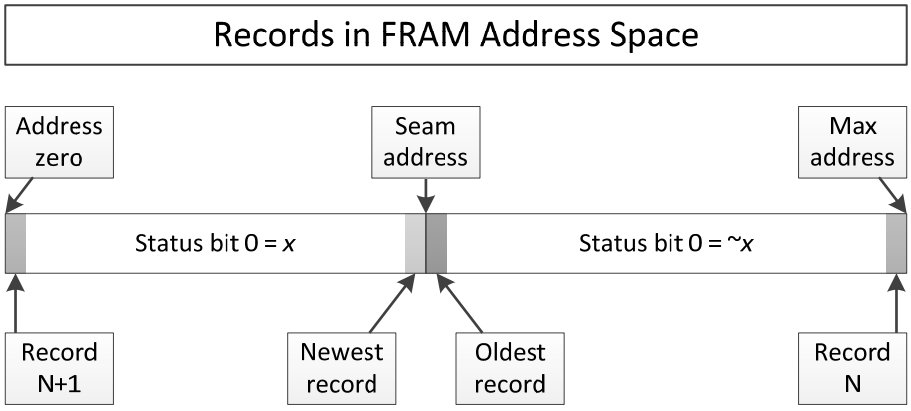
All told, the duration since arm for a given trigger housekeeping structure is:

```
(floor((FRAM address [17..3]*8+7)/10)
 + floor((FRAM_BYTE_CAPACITY-8)/20)*FRAM wraparound count [19..0])
  * (20 ns * (167 cycles + user-defined stall cycles setting))
```

Assuming the default stall cycles setting of 33, one record is sampled every 4000 ns.

FRAM Seam Discovery

The oldest record in FRAM will most likely not be at FRAM address zero. When interpreting data from FRAM, therefore, the records will need to be shifted around such that the oldest record comes first in the data. The address of the oldest record is called the seam address, because it's the point of division between two consecutive cycles through the FRAM circular buffer.



The position of the "seam" in the FRAM circular buffer (where newest and oldest data are adjacent) can be determined by monitoring status bit 0 in each record stored in FRAM. Status bit 0's value is toggled every time the buffer wraps around; starting from the record at address zero, the record in which the value of this bit changes from constant-*x* to constant-not-*x* is the record at the seam address and the oldest record in FRAM. If all records have the same status bit 0 value, then the seam address is zero and the first record is the oldest record.

The seam address can also be derived from the trigger time (which is an FRAM address) and the overlap record count configuration:

(overlap records)(10 FRAM addresses/record) + (stored trigger address) + 10 = (seam address)*

Since the lowest three bits of the trigger time address aren't stored in housekeeping, the stored trigger address needs to be rounded up to the next record boundary, i.e. the next multiple of 10. Alternatively, round up the calculated seam address to the next multiple of 10. The reason for the +10 is that the rest of the formula gives the address of the newest record; to get the address of the oldest record, we must advance the address by one record's worth of address space.

Understanding Configuration Data

The vast majority of the recorder's configuration is stored in nonvolatile memory on the digital board. This memory can be dumped with command *4*, to provide a nearly-complete picture of the recorder configuration on power-up. However, the recorder's currently active configuration may be different from the saved configuration data; multiple commands are needed to retrieve the active configuration.

Active Configuration Data

The current configuration of the unit cannot be dumped in a single command. Multiple commands are required to get a complete copy of the unit's active configuration, and the commands must be issued before the unit is reset or shut off, which will cause the current configuration to be lost. The required commands are:

Table 17. AE Recorder Configuration Command List

e	set/read FRAM position/loops (command 'S' modifies this if not armed or recording)
L	set/read current data-recording address in data memory (block and page)
l	set/read the current duration of the warmup delay, which happens prior to arm-state recording
M	set/read trigger enables
m	set/read trigger threshold enables (both these and the trigger enables from command *M* must be enabled to enable a trigger threshold)
N	set/read overlap records (# of records stored in both FRAM and flash)
n	set/read pages per trigger, max 0x40000 (204 records/page; 250k records/second normally)
o	set/read trigger high/low offsets from quiescent
P	set/read power state (from power board)
p	get data/voltages from power board (see power board docs; valid selections are 1=power-up count, 2=power enable count, 4=capattery voltage, 5=battery voltage, 6=temperature)
Q	set/read between-record delay (acquisition frequency = 50 MHz/(167+this))
q	set/read trigger condition minimum durations
s	get status - current state, arming countdown, time spent waiting before last trigger, etc.
t	set/read multi-trigger enable
V	set/read analog board reference/bias voltages
Y	set/read analog channel offset values
y	set/read analog channel gain values
Z	set/read digital channel threshold voltage
;	enable/disable automatic power-off when recording is finished or end of flash is reached
~	enable/disable 1Hz heartbeat signal output
-	get flash bad block list

Nonvolatile Configuration Data

There are five pages of 128-byte nonvolatile memory used to hold configuration data, each of which contains a different category of data. Each page contains a single data structure, aligned to the start of the page; all bytes between the end of the structure and the end of the page are meaningless. For the sake of forwards compatibility, though, all 640 bytes across all five pages are dumped by command *4*.

When the configuration memory is dumped, all the configuration data it contains is visible, though not very easy to read. Perhaps the best thing to do is to put the data back into the structures in which it's stored, using the following C code. It starts with the definitions of the configuration data structures and ends with the code needed to interpret the dumped data in terms of those data structures.

(Note that the processor in the recorder is little-endian; if you want to use this code on a big-endian machine, reorder the bytes in the multi-byte fields accordingly.)

```
// eNVM types
typedef uint16_t temperature;
typedef uint8_t flag;
typedef uint16_t gain;
```

```
typedef uint16_t offset;

// eNVM constants
#define ENVM_PAGE_SIZE 128
#define ARMSTRING_SIZE 32
#define UNITLABEL_SIZE 32
#define BADBLOCKS_MAXCOUNT 62
#define TRIGGERS_MAXCOUNT 63

// the structures contained in the eNVM pages

// stuff that should be set once then never change
typedef struct eNVM_consts {
	uint16_t badBlockScanHasHappened; // set to 0xbadb when the flash has been scanned for bad blocks
	// bad blocks go here.  How many can there be?  Data sheet says 40 blocks/LUN max can be invalid, and the chip contains 1 LUN.
	uint16_t badBlocks[BADBLOCKS_MAXCOUNT]; // 128 bytes/eNVM page, and the rest of the struct uses 3 bytes; 125 remain.  Divide by 2 to get 62 array entries.
	char unitAddress; // aka unit number.  For a multi-recorder setup, determines what address to listen to.  First command byte is address.
} eNVM_consts;

// mostly for maincontrol
typedef struct eNVM_configuration {
	char unitLabel[UNITLABEL_SIZE];
	uint32_t serialNumber;

	uint32_t delayArmTimeOnStart_seconds; // this is for delay-arm-on-startup, not some sort of default
	uint8_t armOnPower-up; // if this is set, we delay by warmupDelay seconds then ARM immediately
	flag multipleRuns; // if this is set, allow multiple triggers
	uint16_t warmupDelay; // number of seconds to pause for accelerometer warmup in a pseudo-arm state before actually saving data
	uint16_t sampleRate; // units of ksps; min 1, max 299.4 (if you want 300, 299.4 is an error of -0.2%); for human use only
	uint16_t sampleDelayCycles; // cycles of delay between each record-taking; derived directly from sample rate, and actually used by fabric blocks
	uint16_t postTriggerCollectTime; // units of seconds.  Max is 214, so this is a somewhat larger variable than necessary.  For humans only.
	uint32_t postTriggerCollectPages; // directly corresponds to the value in mainCtl
	uint32_t overlapRecords; // how many records are stored in both FRAM and flash
	flag Power-downWhenDone; // whether to turn ourselves off when POWER-DOWN state is reached (if not, goto USER state)
	char armOnPower-upString[ARMSTRING_SIZE]; // what armString gets set to when armOnPower-up activates
	flag outputHeartbeat; // whether to toggle the heartbeat output or just leave it low (such than an attached N-channel MOSFET doesn't conduct)
} eNVM_configuration;

typedef struct eNVM_triggerConfig {
	// the entries in these arrays are ordered by channel index
	uint16_t triggerLevel[NUM_ANALOG_CHANNELS]; // added/subtracted to quiescent measurement to get high/low trigger thresholds
	uint16_t triggerWidth[NUM_ALL_CHANNELS]; // number of consecutive samples the input must satisfy the trigger condition
	flag triggerEnables[NUM_ALL_CHANNELS]; // redundant with trigger directions, but user may prefer ability to retain trigger configuration while disabled
	uint8_t triggerDirections[NUM_ALL_CHANNELS]; // high, low, both, or neither
} eNVM_triggerConfig;

typedef struct eNVM_armState {
	uint32_t delayArmTimeUsed_seconds; // the value passed in with the "delay arm" command; requirement wants minutes, that's a job for the GUI
	char armString[ARMSTRING_SIZE]; // aka "arming time stamp" or maybe "event description"; provided with arming command
	temperature t1Temp_arm; // 't1' is the arming 'trigger', i.e. the arm command in whatever form it takes
	uint16_t memoryOverwriteProtect; // memory protection must be disabled in the configuration utility, but this also gets written during arm
```

```
} eNVM_armState;

// probably clear these out (or at least triggerCount) whenever arm-lock is set
typedef struct eNVM_trigState {
        temperature t2Temps_trigger[TRIGGERS_MAXCOUNT]; // 't2' is the trigger as we know it
        uint16_t triggerCount; // triggers detected, thus also the number of valid entries in the
above table; if greater than TRIGGERS_MAXCOUNT, all entries in the above table are valid
} eNVM_trigState;

// envmDump is a 640-byte buffer containing the output of command 4, converted from characters
back to bytes

eNVM_consts *ENVM_CONSTANTS_PTR = (eNVM_consts*)(envmDump);
eNVM_configuration *ENVM_CONFIGURATION_PTR = (eNVM_configuration*)(envmDump + ENVM_PAGE_SIZE);
eNVM_triggerConfig *ENVM_TRIGGERCONFIG_PTR = (eNVM_triggerConfig*)(envmDump + ENVM_PAGE_SIZE *
2);
eNVM_armState *ENVM_ARMSTATE_PTR = (eNVM_armState*)(envmDump + ENVM_PAGE_SIZE * 3);
eNVM_trigState *ENVM_TRIGSTATE_PTR = (eNVM_trigState*)(envmDump + ENVM_PAGE_SIZE * 4);


// at this point, you can access the contents of the structures like so:
printf("Unit label: ");
fwrite(ENVM_CONFIGURATION_PTR->unitLabel, 1, UNITLABEL_SIZE, stdout);
printf("\n");
printf("Serial number: %u\n", ENVM_CONFIGURATION_PTR->serialNumber);
```

Warning: The fields in the structures may be “optimized” by your compiler into a different sequence or packing. It is technically possible to copy this code verbatim and have fields not be populated properly. Should this happen, it would probably be simplest to parse out the individual fields one at a time from the data returned by command *4*; the fields in each page dumped by that command will be in the order given above, each structure field in sequence from top to bottom.

On-Board Instrumentation Support Electronics Design

For earth penetrator applications, the AE Recorder modules require only a battery. The module is complete with a power interface board to connect the battery when commanded. For a sled track application that must be self-contained while also maintaining electrical isolation between the instrumentation and the outside world, additional support electronics are required to communicate with the system and manage the battery. Three additional circuit boards handle these functions: the 3A7001 Safe-State Monitor with protected electrical connections for battery maintenance and instrumentation status; 3A7002 External Fiber-Optic Interface to provide electrically-isolated communication and control; and the 3A7004 Battery Management System board.

3A7001 Safe-State Monitor Board

This simple board includes a 15-pin connection to a matching connector on the 3A7002 board, with schematic in Figure 31 and a blank board shown in Figure 32. A circular, 10-pin Glenair 803-005-7-10 socket connector with metal dust cap provides access to the necessary signals while still meeting isolation requirements. Diodes D1 and D2 are bright LEDs providing redundant indication that the instrumentation is powered. The remaining 9-pin connector supports two loops of serially-connected LEDs. From one to six LEDs can be used in each string. A 40mA drive from the 3A7002 board pulses the LEDs at a programmable rate to allow external synchronization of photometric equipment with the AE Recorder memory.

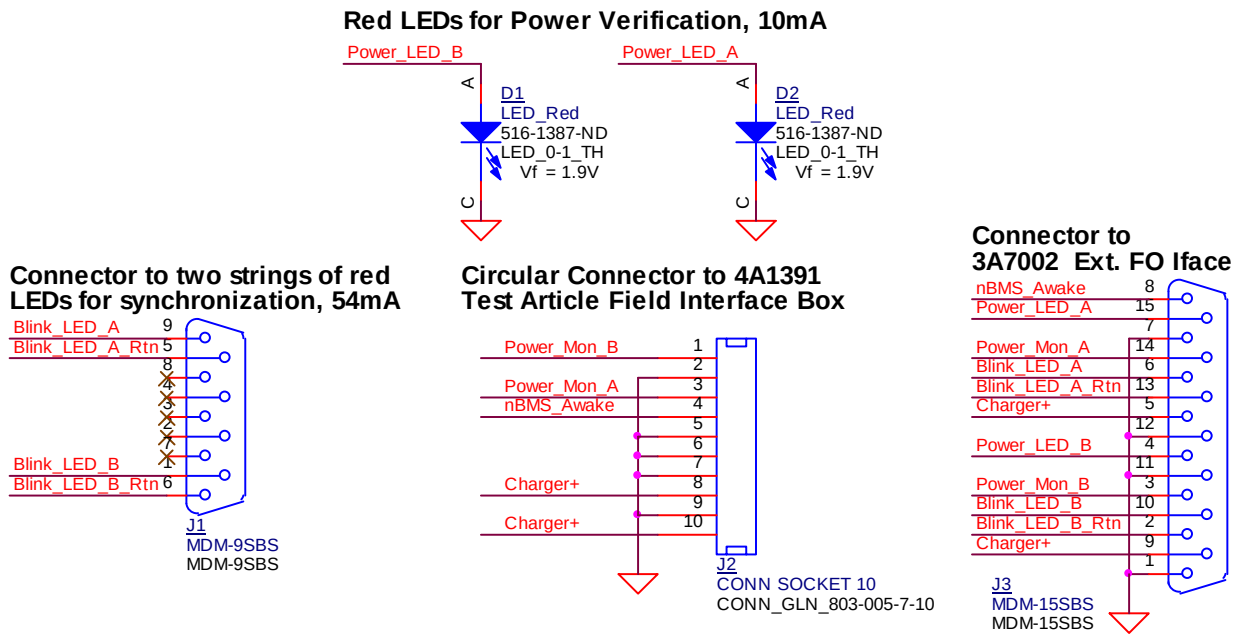


Figure 31. 3A7001-002 Safe-State Monitor Board Schematic

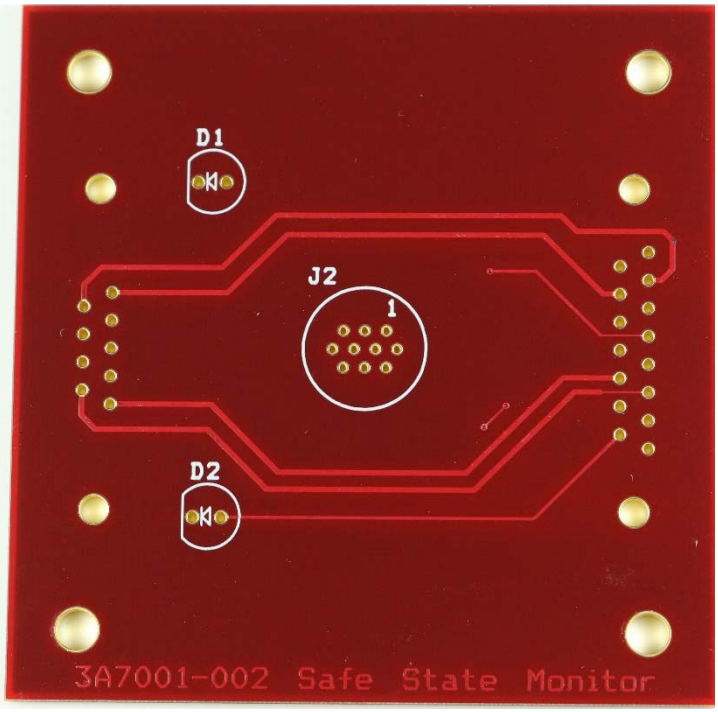


Figure 32. 3A7001-002 Safe-State Monitor Board

3A7002 External Fiber-Optic Interface Board

Multi-mode fiber from the sled track connects to the test article through the 3A7002-003 External Fiber-Optic Interface Board, which has two fiber photodiodes, four receivers, and two transmitters. Multi-mode fibers show greater resistance to failure when exposed to the usually dust-filled environment of a sled track test. The two fiber photodiodes provide redundant power-on detectors, driving the gates of transistors to control application of battery power to the entire

system. Two each of the four receiver fiber interfaces drive the Trigger input shared among all the AE Recorder module. The other two fiber receivers are employed for the UART serial interface. Both redundant serial interface receive signals are monitored by the board's Microchip PIC24FJ64GB204 microcontroller. If the microcontroller detects that the default channel does not have data while the back-up fiber does, communication is automatically switched to the active fiber for all the AE Recorders. The two fiber transmitters handle the transmit part of the UART serial interface, and both redundant transmit fibers are always active.

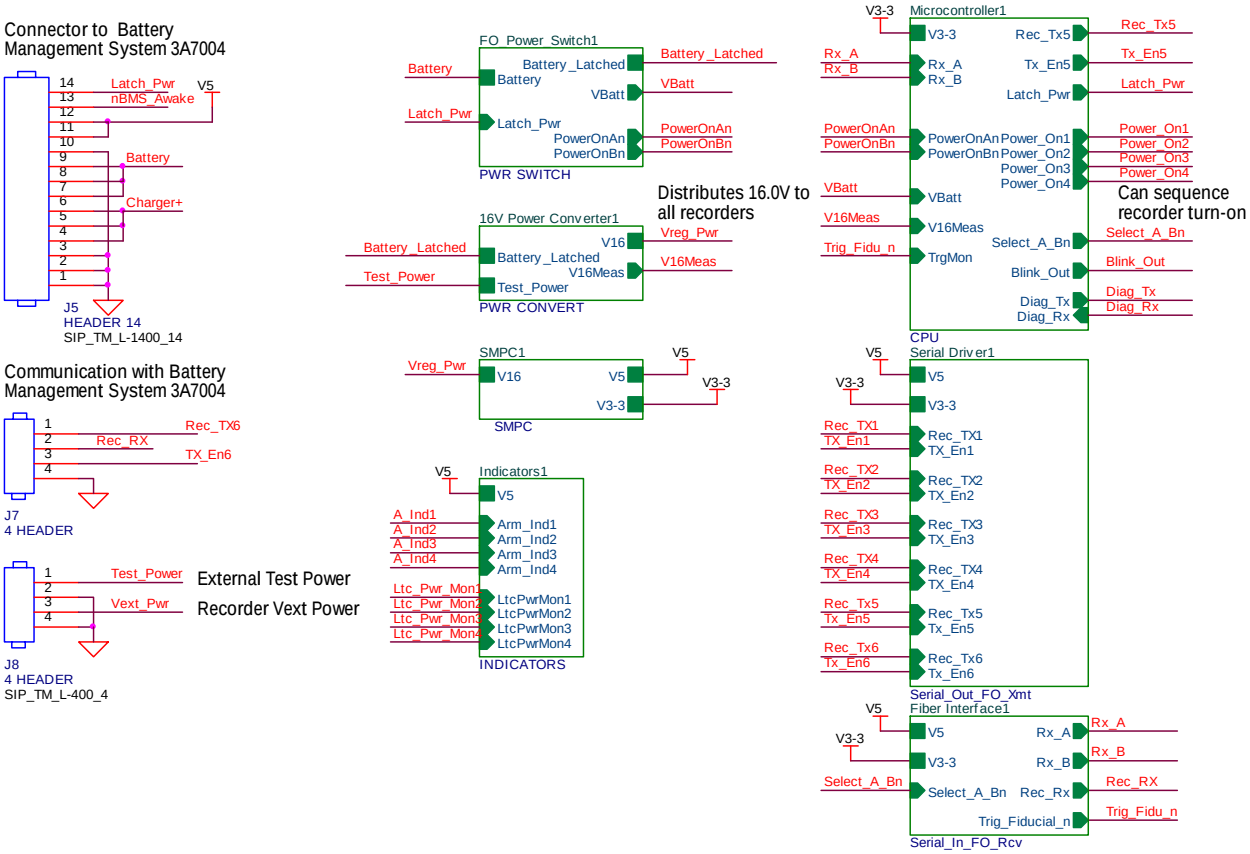


Figure 33. Fiber-Optic and Microcontroller Schematic Portion of the 3A7002 Board

In addition to monitoring the fiber-based serial communication, the microcontroller firmware also qualifies the power-on signal from the photodiode outputs. The signal must remain asserted for more than one second to qualify powering the on-board instrumentation. (This one-second qualification was omitted from the Impact 1 test firmware.) Following detection, qualification, and latching power on, the 3A7002 waits for commands from the serial interface to power the AE Recorder modules. Each recorder is powered using a separate command. All of the fiber interface modules are shown in the top-level hierarchical schematic in Figure 33.

Finally, the microcontroller also generates a "Blink" synchronization signal shared among all the AE Recorder modules and output to the 3A7001 Safe-State Monitor board. This Blink signal will appear in each of the AE Recorder modules' data, and when connected via the 3A7001 board to an externally visible LED string, can also synchronize AE Recorder data with external

photometric measurements. The period and duty cycle of the Blink signal is field-configurable using the serial command interface.

The Blink feature was a late add-on for the previous test, Impact 1, and had some implementation flaws. First, the frequency and period were fixed in the firmware without an ability to field configure. This inflexibility made debugging and testing by the photometric crew more difficult. And most importantly, the Blink signal was previously routed to Discrete signal inputs on each AE Recorder module, which required cable connections soldered during test article assembly. In the revised hardware, this latter problem was corrected by a change to the AE Recorder that included an optional input signal in the 15-pin power and interface connector, the "Heart_Sync_n" signal shown in Figure 35. This avoided the need for field connections because the signal was directly connected to the recorder. The Discrete byte now includes Trigger, Blink Synchronization, and Discrete 1 through 6. The signals are routed to the 3A7001 Safe-State Monitor board from connector J12, shown in Figure 34. Constant-current diodes limit power from the 16V Vreg_Pwr that is then applied to the interface. Three, 18mA current-limit diodes in parallel drive the Blink LED strings to produce a bright photometric synchronization signal.

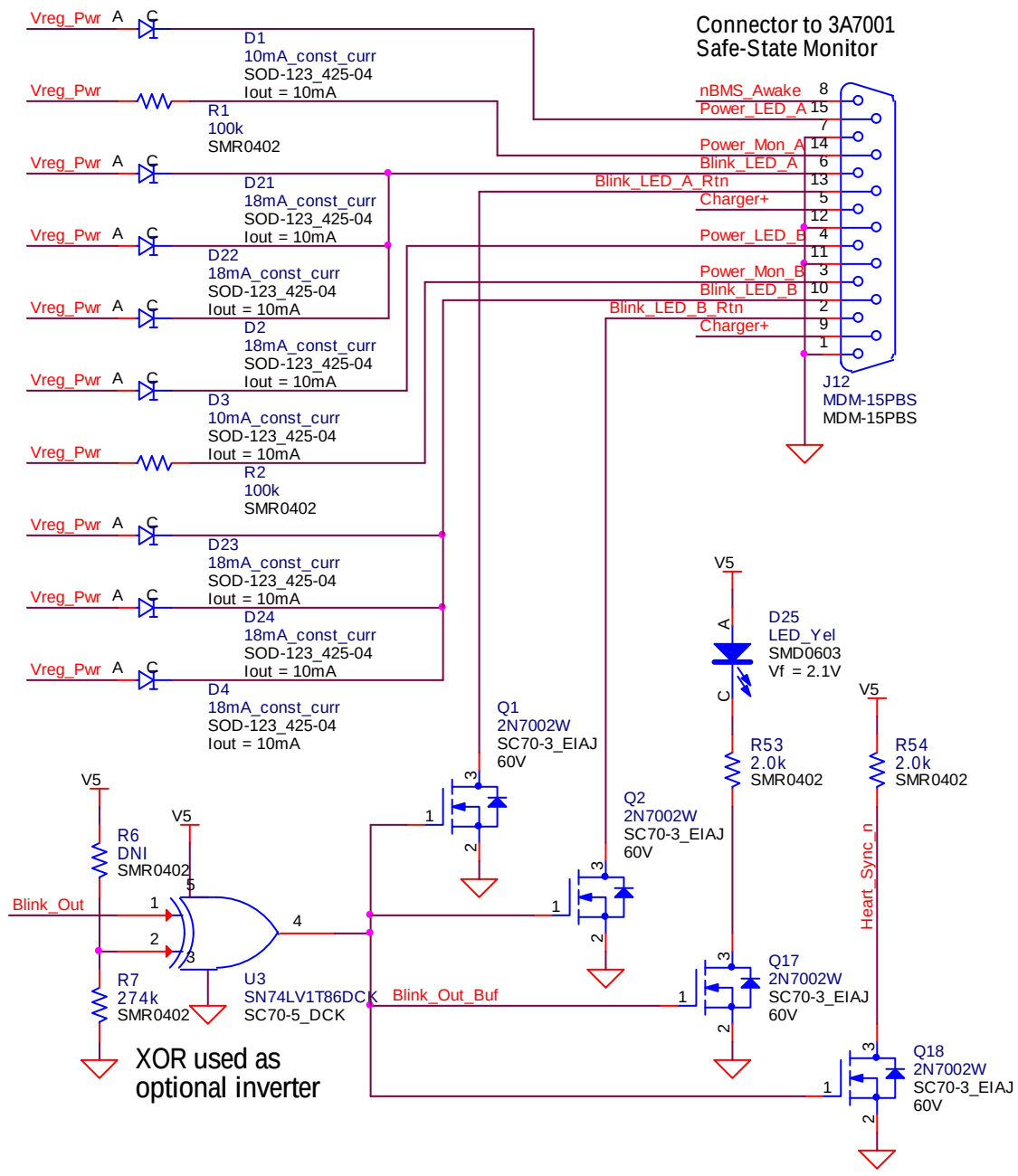


Figure 34. Connections to the 3A7001 Safe-State Monitor Board

Connections to the AE Recorder modules are shown in Figure 35. Each recorder transmits data on a separate line, Rec_Tx1 through Rec_Tx4, with the signals combined on the 3A7002 board using the corresponding TX_En1 through TX_En4 signals.

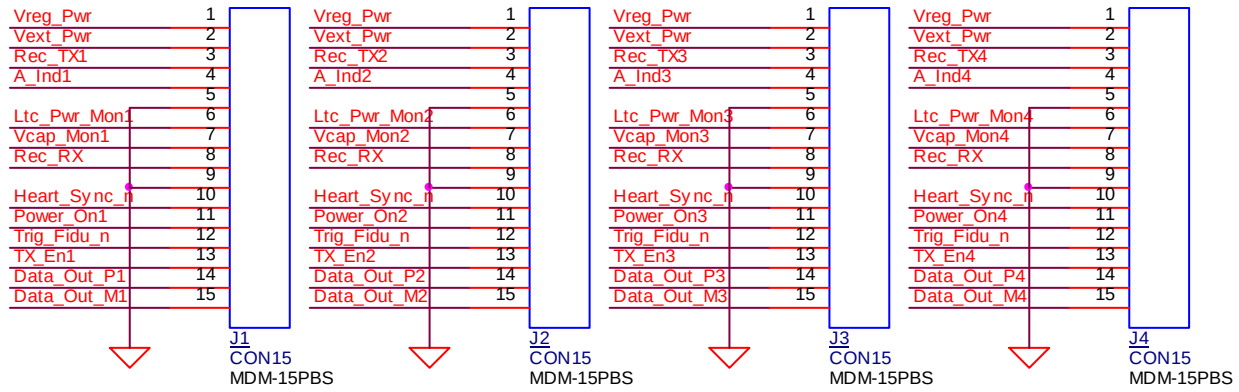


Figure 35. AE Recorder Connections from the 3A7002 Board

Power on the 3A7002-003 External Fiber-Optic Interface Board is first stepped-down from the raw 36.5V maximum $LiFePO_4$ battery voltage to 16V using the Switch-Mode Power Converter shown in Figure 36. The 16V is distributed to all the AE Recorder modules to charge each recorder's internal capattery. Current draw for all four fully-powered AE Recorder modules plus the support electronics is about 650mA, well within the SMPC's 2A capability. This SMPC is about 80% efficient, so draws less than 0.5A from the 32V nominal battery. Another SMPC, the same low-noise Linear Technology LTM8031 module used in the AE Recorders, steps down the 16V to 5V for use by the fiber optics and microprocessor.

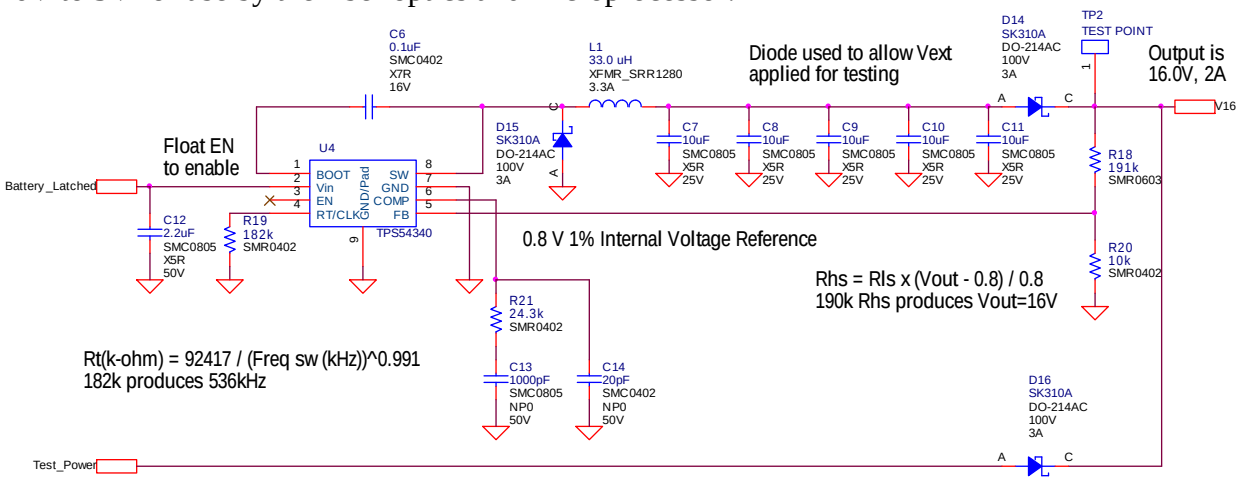


Figure 36. SMPC Produces 16V Power from the 32V Nominal Battery

3A7004 Battery Management System

Lithium-ion cells have the highest energy density by volume and weight, and were used in the AE Recorder on-board instrumentation system because the volume available for a battery was a design constraint. The trade-off using a lower-density chemistry would have been shorter operating time and a requirement to top-off the battery immediately before the test. However, if lithium chemistry cells are used outside their safe operating area, the cells can fail catastrophically by bursting into flames or exploding. A critical factor in safe operation is ensuring cells are not over-charged, and to a lesser extent, over-discharged. The Battery Management System board monitors cell voltages during charge, allowing microcontroller firmware to apply balancing loads to charge all cells to equal capacity without over-charging. A fully-charged pack with all cells having balanced capacity also helps avoid over-discharge on

cells that might otherwise have received an incomplete charge. The BMS disconnects the charger current when the pack is fully charged, and when the battery pack is tested following assembly, the BMS also disconnects the load during discharge cycling. The load disconnect feature is disabled for test operations. The BMS continuously reports cell data during charging using the fiber-optic serial interface on the 3A7002 board.

The AE Recorder 32V nominal on-board instrumentation battery is composed of 10 each K2 Energy LFP26650P80 cells, 2.6A-Hr nominal. Since the instrumentation provides a 0.5A load, a fully-charged battery should operate the system for about 5 hours. The very low self-discharge rate from $LiFePO_4$ and the associated battery management system should allow recharging after test article assembly and check-out is complete and no further servicing for several months, however, since the recharging capability is readily available, the battery is topped off on the day of test. During non-operational time, the battery management system is set to Sleep mode which draws about 0.1mA, equivalent to less than 5% self-discharge after one month.

The 3A7004-002 Battery Management System board uses two Texas Instruments bq76PL536a BMS devices connected in series, with each chip monitoring 5 of the 10 cells. The top level of the hierarchical schematic is shown in Figure 37. Although the bq76PL536a battery management system chip gets power directly from the battery, the microcontroller and similar circuits on the 3A7004 board need 5V power. This is provided from the 3A7002 External Fiber-Optic Interface board. Once the power-on photodiode is illuminated from the optical fiber, the 3A7002 board is powered and in turn powers the 3A7004 board.

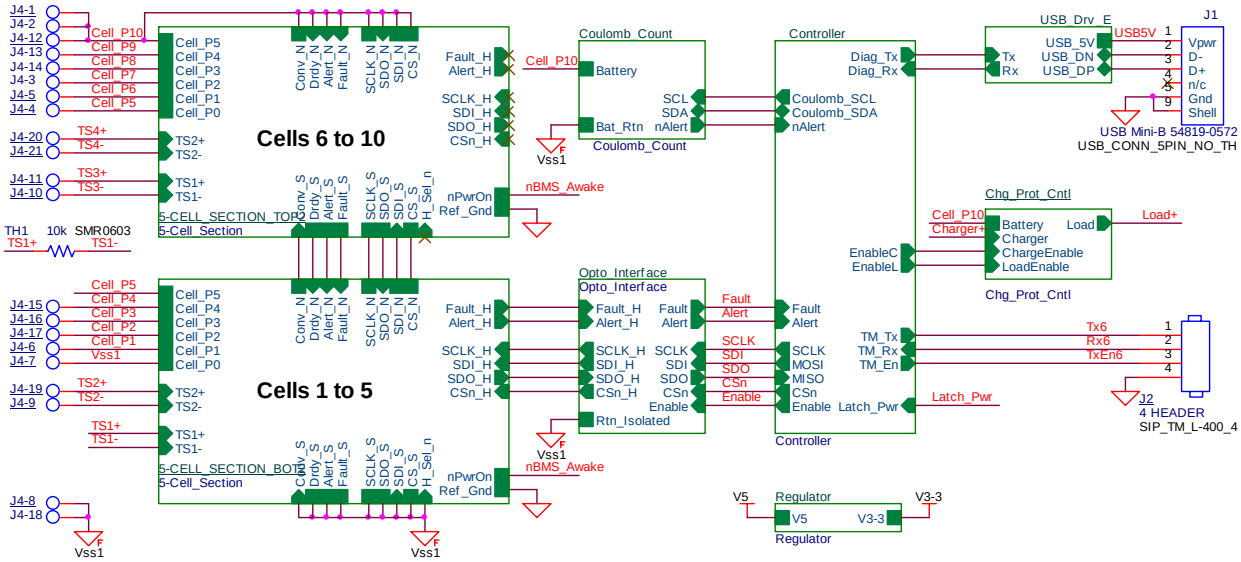


Figure 37. Battery Management System Hierarchical Schematic Top Level

One problem encountered during the Impact 1 test article assembly was that the BMS was inadvertently left operating. In Sleep mode the BMS consumes only 0.1mA, but when awake the power climbs to 12mA. This can discharge the 2.6A-Hr battery in about nine days. To make this condition detectable without exposing raw battery voltage outside the instrumentation, an Opto-MOS switch was incorporated in the design. The nBMS_Awake signal is an open-drain output from a TLP3250 Opto-MOS transistor. When in Sleep mode, the bq76PL536a chip

removes Reg50 power and the output will be open. When the BMS is active, it will pull the signal to ground. A resistance measurement between nBMS_Awake (labeled BMS AWAKE on the test article interface) and Return should show less than an Ohm or so when the BMS is active.

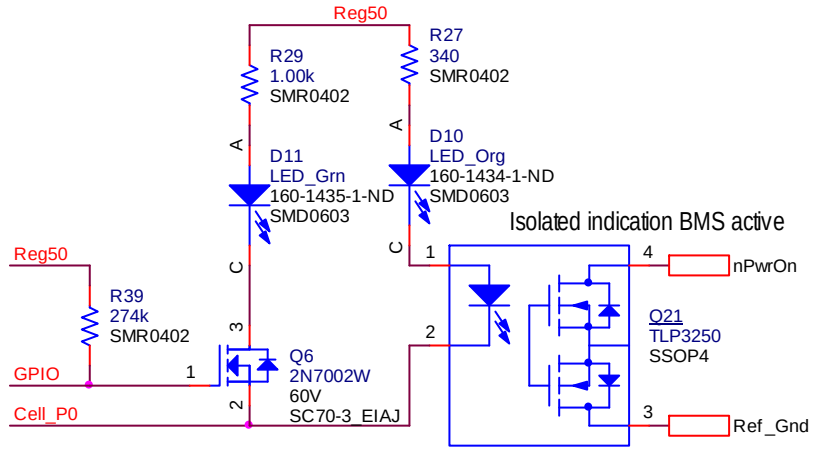


Figure 38. Battery Management System Sleep Monitor Circuit

The assembled 3A7004 board with an MDM-21S harness installed is shown in Figure 39. The 10-cell battery pack is built with a mating MDM connector to attach individual cells to the balancing circuit and the main battery output and return. Spacing between the circuit board edge and the battery pack housing is tight, so that a board-mounted MDM connector was useless. With no need to route all connections to a single point, a harness was used with wires connected close to their associated circuits, and only four board layers were needed.

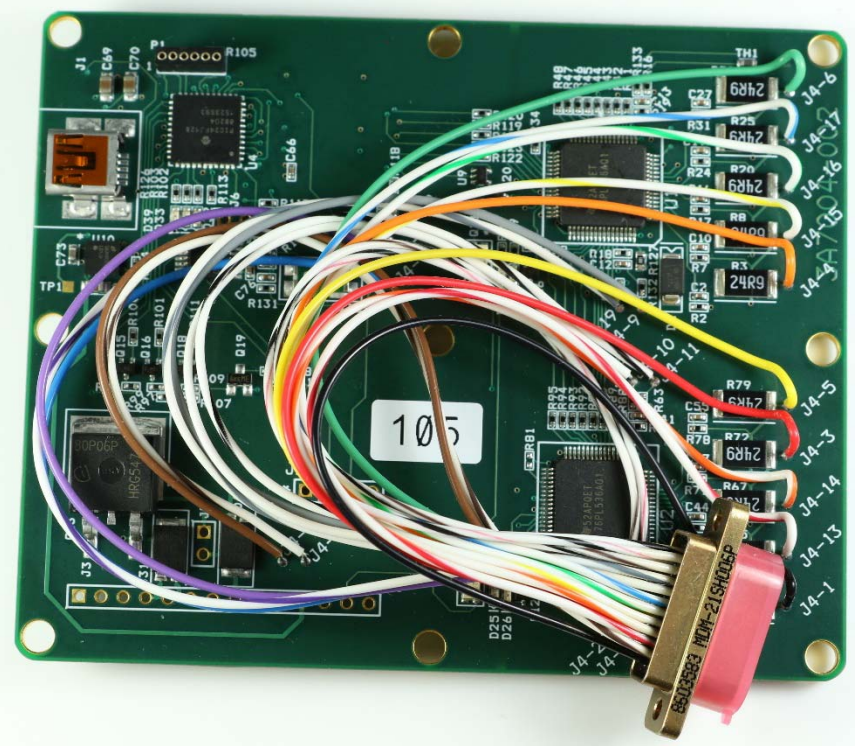


Figure 39. 3A7004-002 Battery Management System Board

The right half of the board has the bq76PL536a chips and the far right edge the balancing resistors. The upper left quadrant holds the supervisory microcontroller, a Microchip PIC24FJ128GB204. This is essentially the same microcontroller used on the AE Recorder's 3A6999 Power / Interface and 3A7000 Analog boards, except in a 44-pin package. The USB connector near the microcontroller is helpful for firmware development. The UART serial interface for reporting data during sled track operations connects through J2 to the matching connector J7 on the 3A7002 board. The battery output and the 5V input appear on J3, which is shown on the bottom, left edge of the 3A7004 board. The pin definition in J3 matches J5 on the 3A7002 board, which is physically adjacent to the 3A7004 board.

CONCLUSIONS

All four Anomalous Environment Recorder modules installed in the B61-12 sled track test article successfully returned all data during the first use on a March 2016 test, Impact 1. The design, with some minor updates based on this test experience, will form the on-board instrumentation for two subsequent rocket sled tests. The AE Recorder also provides an expanded capability, high fidelity, robust data recorder for earth penetrator applications.

This first use of a high-speed serial interface in penetrator instrumentation makes the extreme depth of memory practical, with data extraction time about the same as the recording time. The combination of both FRAM and Flash non-volatile memory was effective to capture the required pre-trigger and post-trigger data.

Shock levels may not have been high enough to qualify the design for earth penetrator applications; in particular the robustness of one component is of concern: the switch-mode power converter whose efficiency helped enable the long, 5-second operating time after external power loss and reduced power requirements by 50%. However, since the AE Recorder uses proven shock-hardened mechanical design and encapsulation methods, the design is expected to handle earth penetrator shock levels including the high jerk.

REFERENCES

1. *SAND2009-6461C, A Review of the Sandia Rocket Sled Track Accident.* **May, Rodney A.** Kansas City, MO : s.n., 2009 . Western Regional Strain Gage Committee Test & Measurement Conference.

2. **Medina, AJ, Stofleth, JH, Dinallo, MA.** *SAND2009-0180, Technical Advisory Team (TAT) Report on the Rocket Sled Test Accident of October 9, 2008.* s.l. : Sandia National Laboratories, January 2009.

3. **Partridge, ME, Larsen, CA, McGrogan, DP, Neidigk, M.** *SAND2013-0391 3DDR-AM Brass-Board Development.* s.l. : Sandia National Laboratories, 2013.

4. **Welch, B and Partridge, M.** *Methods for Automatic Trigger Threshold Adjustment. 8,676,540* United States of America, March 18, 2014.

5. **Partridge, Michael E and Welch, Benjamin J.** *SAND2005-6681, HiCapPen Earth Penetrator Instrumentation Development.* 2005.

6. **Adolf, D. B., M. Neidigk, R. Chambers, M. Neilsen, S. Spangler, K. Austin.** *Packaging Strategies for PCB Components, Vol. I: Materials & Thermal Stresses, Report SAND2011-4751.* s.l. : Sandia National Laboratories, 2011.

7. **Curtis, S., Neidigk, M., & O'Malley, P.** *SAND2013-1668, Electronics Packaging for High-G Environments: Literature Survey, .* s.l. : Sandia National Laboratories, March 2013.

8. **AVX Corporation.** *AVX BestCap® Ultra-low ESR High Power Pulse Supercapacitors Version 11.4 data sheet.* 2011.

9. **Zetex.** *ZDS1009 SM-8 Complementary Current Mirror data sheet.* 2000.

10. **Meggitt.** *Endevco Model 7270A Piezoresistive Accelerometer.* 2011.

11. **Irwin, J. D.** *The industrial electronics handbook. The electrical engineering handbook series.* s.l. : CRC Press, 1997.

12. **Piersol, A., Paez, T. and Harris, C.** *Harris' shock and vibration handbook.* s.l. : McGraw-Hill, 2009.

13. **Microchip.** *PIC24FJ64GB004 Family Data Sheet.* 2010.

14. **Micron Technology.** *MT29F8G08ABABA 8Gb Asynchronous/Synchronous NAND Flash Memory Features data sheet.* 2009.

15. **Ramtron International.** *FM23MLD16 8Mbit F-RAM Memory data sheet.* 2008.

16. **Hoke, Darren; Partridge, Michael E; Mittas, Anthony; Henry, Edward A; Lockhart, Randall R.** *SAND2002-3576, Micro High-g Acceleration Recorder LDRD Final Report. (Note: This report refers to the 3AMP development.).* 2002.

17. **Partridge, Michael E and McGrogan, David P.** *SAND2012-0719, 3DDR-AM Breadboard Status Report.* 2012.

18. **Texas Instruments.** *ADS7865 Dual, 12-Bit, 3+3 or 2+2 Channel, Simultaneous Sampling Analog-to-Digital Converter Data Sheet.* 2010.

19. **Tektronix, Inc.** *Application Note, Effective Bits Testing Evaluates Dynamic Performance of Digitizing Instruments.* s.l. : Tektronix, 2008.

20. **Partridge, Michael E.** *SAND2011-4614P, 3DDR-AM Block 2 Requirements.* 2011.

21. **Linear Technology.** *LTM8031 Ultralow Noise EMC 36V, 1A DC/DC µModule Regulator data sheet.* 2009.

22. **Texas Instruments .** *TPS54620 4.5V to 17V Input, 6A Synchronous Step Down SWIFT™ Converter data sheet.* 2011.

23. **Larsen, Cory A.** *SAND2011-9467, Signal Conditioning Circuitry Design for Instrumentation Systems.* 2011.

24. **Wang, Darren and Niemczura, Johnathan.** *SAND2012-0542, DTRA/SNL Fuze Diagnostic Recorder Inductor Testing: FY11 Year End Report.* 2012.
25. **Analog Devices.** *AD8224 Precision, Dual-Channel, JFET Input, Rail-to-Rail Instrumentation Amplifier Revision B data sheet.* 2010.
26. **Microsemi Corporation.** *SmartFusion Customizable System-on-Chip (cSoC) Revision 7 data sheet.* 2011.
27. **Silicon Labs.** *Si500S Single-Ended Output Silicon Oscillator Rev. 1.0 data sheet.* 2011.
28. **Micro Oscillator Inc.** *MOI-2000 CMOS Clock Oscillator data sheet.* 2008.
29. **Actel Corporation.** *SmartFusion Development Kit User's Guide.* 2011.
30. **Texas Instruments.** *PCA9557 Remote 8-Bit I2C and SMBus Low-Power I/O Expander with Reset and Configuration Registers data sheet.* 2008.
31. **Sandia National Laboratories.** *ADAGIO/ANDANTE Users Reference Manual.* Albuquerque, New Mexico : s.n., 2005.
32. *A simplified potential energy clock model for glassy polymers.* **Adolf, D. B., R. S. Chambers, and M. A. Neidigk.** Polymer, 2009, Vols. 50, pp. 4257-4269.
33. **Scherzinger, W. M., and D. C. Hammerand.** *Constitutive Models in Lame Report SAND2007-5873 .* s.l. : Sandia National Laboratories, 2007.
34. **Sandia National Laboratories.** *PRESTO Users Reference Manual.* Albuquerque, New Mexico : s.n., 2005.
35. **Walter, P. L.** *"Measurement systems engineering." Short Course, Taught at Sandia National Laboratory.* 2010.
36. **Jung, W. G.** *Op-Amp Applications Handbook.* s.l. : Newnes, 2005.
37. **Texas Instruments.** *FilterPro User's Guide.* 2011.
38. **Bishop, J.** *Filter-pro low-pass design tool.* s.l. : Texas Instruments, 2002.
39. **Stitt, R. M.** *Voltage-reference filters, Application Note SBVA002.* s.l. : Texas Instruments, 1991.
40. **Ryerson, D. E.** *Signal conditioning primer.* s.l. : Sandia National Laboratories, 1996.
41. —. *Distortion of penetrator data by analog and digital filters.* s.l. : Sandia National Laboratories, 1989.
42. **Karki, J.** *Active low-pass filter design.* s.l. : Texas Instruments, 2000.
43. **Oljaca, M.** Understand the limits of your ADC input circuit before starting conversions. *Analog Zone.*
44. **Paarmann, L. D.** *Design and Analysis of Analog Filters: A Signal Processing Perspective.* s.l. : Kluwer Academic Publishers, 2001.
45. **Oppenheim, A. V., Schafer, R. W. and Buck, J. R.** *Discrete Time Signal Processing.* s.l. : Prentice Hal, 1999.
46. *The digital all-pass filter: A versatile signal processing building block.* **Regalia, P.A., Mitra, S. K. and Vaidyanathan, P.** no. 1, s.l. : Proceedings of the IEEE, 1988, Vol. vol. 76.
47. **Texas Instruments.** *Analysis of the Sallen-Key architecture.* 2002.
48. **Steffes, M.** *Design methodology for MFB filters in ADC interface applications.* s.l. : Texas Instruments, 2006.
49. *New concept of delay equalized low-pass Butterworth filters.* **Piskorowski, J. and Kaszynski, R.** vol. 1, pp. 171-175, 2006, Vol. 2006 IEEE International Symposium on Industrial Electronics.
50. **Zumbahlen, H., ed.** *Linear Circuit Design Handbook, ch. 8: Analog Filters.* s.l. : Newnes, 2008.

51. **Oljaca, M., and Downs, R.** Designing SAR ADC drive circuitry – part I: A detailed look at SAR ADC operation. *Analog Zone.*
52. —. Desiging SAR ADC drive circuitry – part III: Designing the optimal input drive circuit for SAR ADCs. *Analog Zone.*
53. —. Designing SAR ADC drive circuitry – part II: Input behavior of SAR ADCs. *Analog Zone.*
54. **Mancini, R. and Carter, B., eds.** *Op Amps for Everyone – Third Edition.* s.l. : Newnes, 2009.
55. **Analog Devices.** *2.7 V to 5.5 V, 450 µA, Rail-to-Rail Output, Quad, 12-/16-Bit nanoDACs.* 2006.
56. **Texas Instruments.** *PGA112 Zero-Drift, Programmable Gain Amplifier with Mux.* 2008.
57. **Oljaca, M., and Downs, R.** Designing SAR ADC drive circuitry – part II: Input behavior of SAR ADCs. *Analog Zone.*
58. **Texas Instruments.** *SN65LV1224B 10-MHz To 66-MHz, 10:1 LVDS Serializer / Deserializer.* 2009.
59. *SAND2010-6971C, Premature ignition of a rocket motor.* **Moore, Darlene Ruth.** s.l. : U.S. Dept. of Energy, Office of Scientific and Technical Information, 2010.
60. **Neidigk, M., Starr, M., & Curtis, S.** *SAND2015-5311, Packaging Strategies for Harsh Thermal and Mechanical Environments Volume III: Assembly Studies.* s.l. : Sandia National Laboratories, July 2015.

APPENDIX A – ALPHABETICAL COMMAND LIST

1	print the entire contents of SRAM (4096 bytes), followed by a 4-byte CRC checksum; only the first 4080 bytes contain record data
2	print a page of FRAM (4096 bytes), followed by a 4-byte CRC checksum; the beginning of the page is not guaranteed to line up with the beginning of a record
3	print a page of flash (4182 bytes), followed by a 4-byte CRC checksum; each group of 12 records (one glob) is followed by 6 bytes of Reed-Solomon error-correcting code (ECC)
4	print the raw contents of the eNVM configuration memory (640 bytes)
A	arm the recorder immediately; the recorded arm delay is zero
B	clear FRAM
b	clear flash
C	force trigger
D	arm after a delay
d	read current time until arming happens
E	re-enable arming/data taking (clear the memory lock in eNVM) if not taking data or counting down a timer
e	set/read FRAM position/loops (command 'S' modifies this if not armed or recording)
F	dump FRAM page over SERDES (optional: dump N pages)
f	force-stop all delaying/arming operations; recording won't be stopped w/o reset (cmd *R*)
G	dump flash page over SERDES (optional: dump N pages)
g	dump the next page (FRAM or flash) over SERDES (optional: dump next N pages)
H	put test patterns in FRAM (disabled if memory is locked)
h	put test patterns in flash (disabled if memory is locked)
I	send raw i2c data
i	receive raw i2c data
J	enter manual menus
j	enable/disable debug messages
K	retry the most recent SERDES dump from FRAM
k	retry the most recent SERDES dump from flash
L	set/read current data-recording address in data memory (block and page)
l	set/read the current duration of the warmup delay, which happens prior to arm-state recording
M	set/read trigger enables
m	set/read trigger threshold enables (both these and the trigger enables from command *M* must be enabled to enable a trigger threshold)
N	set/read overlap records (# of records stored in both FRAM and flash)
n	set/read pages per trigger, max 0x40000 (204 records/page; 250k records/second normally)
O	set/read SERDES enable state
o	set/read trigger high/low offsets from quiescent
P	set/read power state (from power board)
p	get data/voltages from power board (see power board docs; valid selections are 1=power-up count, 2=power enable count, 4=capattery voltage, 5=battery voltage, 6=temperature)

Q	set/read between-record delay (acquisition frequency = 50 MHz/(167+this))
q	set/read trigger condition minimum durations
R	reset/reboot the microcontroller and flash
S	read current voltages/inputs
s	get status - current state, arming countdown, time spent waiting before last trigger, etc.
T	hardware test (verify connections to FRAM, flash)
t	set/read multi-trigger enable
U	set/read unit serial number (9 digits max)
u	set/read the description string of the unit
V	set/read analog board reference/bias voltages
v	set/read arm-on-power-up
W	get current mode: U=User, D=Delay-Arm, W=Warmup, F=Arm-First, R=Recording, A=Arm-Again, P=Power-down, L=Locked (can't arm, data present), O=other
w	get unit identity and firmware revision
X	save/revert configuration data to values stored in nonvolatile eNVM
Y	set/read analog channel offset values
y	set/read analog channel gain values
Z	set/read digital channel threshold voltage
z	read arm string
;	enable/disable automatic power-off when recording is finished or end of flash is reached
~	enable/disable 1Hz heartbeat signal output
+	set/read unit address; value is ASCII hex equivalent of address character, e.g. '1' = 0x31
=	get number of usable flash pages (0x40000 if no bad blocks)
_	get number of pages of recorded data (if 0, no data is present; if same as '=', flash is full)
-	get flash bad block list
?	print a help message and enable input echo, for simpler direct human control
/	disable input echo, for returning to program-controlled operation after using command “?”

APPENDIX B – COMMAND LIST BY CATEGORY

Collecting Data

Preparing to Arm

E	re-enable arming/data taking (clear the memory lock in eNVM) if not taking data or counting down a timer
B	clear FRAM
b	clear flash

Arming

A	arm the recorder immediately; the recorded arm delay is zero
D	arm after a delay
v	set/read arm-on-power-up

While arming or armed

d	read current time until arming happens
f	force-stop all delaying/arming operations; recording won't be stopped w/o reset (cmd *R*)
C	force trigger

Retrieving Data

Via SERDES (quickly)

F	dump FRAM page over SERDES (optional: dump N pages)
G	dump flash page over SERDES (optional: dump N pages)
g	dump the next page (FRAM or flash) over SERDES (optional: dump next N pages)
K	retry the most recent SERDES dump from FRAM
k	retry the most recent SERDES dump from flash
O	set/read SERDES enable state

Via serial (slowly)

1	print the entire contents of SRAM (4096 bytes), followed by a 4-byte CRC checksum; only the first 4080 bytes contain record data
2	print a page of FRAM (4096 bytes), followed by a 4-byte CRC checksum; the beginning of the page is not guaranteed to line up with the beginning of a record
3	print a page of flash (4182 bytes), followed by a 4-byte CRC checksum; each group of 12 records (one glob) is followed by 6 bytes of Reed-Solomon error-correcting code (ECC)
4	print the raw contents of the eNVM configuration memory (640 bytes)

Configuration

Saving and reverting settings

X	save/revert configuration data to values stored in nonvolatile eNVM

Recording	
l	set/read the current duration of the warmup delay, which happens prior to arm-state recording
N	set/read overlap records (# of records stored in both FRAM and flash)
n	set/read pages per trigger, max 0x40000 (204 records/page; 250k records/second normally)
Q	set/read between-record delay (acquisition frequency = 50 MHz/(167+this))
t	set/read multi-trigger enable
;	enable/disable automatic power-off when recording is finished or end of flash is reached
Triggers	
M	set/read trigger enables
m	set/read trigger threshold enables (both these and the trigger enables from command *M* must be enabled to enable a trigger threshold)
o	set/read trigger high/low offsets from quiescent
q	set/read trigger condition minimum durations
Analog board	
V	set/read analog board reference/bias voltages
Y	set/read analog channel offset values
y	set/read analog channel gain values
Z	set/read digital channel threshold voltage
Outputs	
~	enable/disable 1Hz heartbeat signal output
Saving and reverting settings – again, because it's important!	
X	save/revert configuration data to values stored in nonvolatile eNVM

Diagnostics

d	read current time until arming happens
p	get data/voltages from power board (see power board docs; valid selections are 1=power-up count, 2=power enable count, 4=capattery voltage, 5=battery voltage, 6=temperature)
S	read current voltages/inputs
s	get status - current state, arming countdown, time spent waiting before last trigger, etc.
T	hardware test (verify connections to FRAM, flash)
W	get current mode: U=User, D=Delay-Arm, W=Warmup, F=Arm-First, R=Recording, A=Arm-Again, P=Power-down, L=Locked (can't arm, data present), O=other
w	get unit identity and firmware revision
z	read arm string
=	get number of usable flash pages (0x40000 if no bad blocks)
–	get number of pages of recorded data (if 0, no data is present; if same as '=', flash is full)
-	get flash bad block list

Recorder Identity

U	set/read unit serial number (9 digits max)
u	set/read the description string of the unit
w	get unit identity and firmware revision
+	set/read unit address; value is ASCII hex equivalent of address character, e.g. '1' = 0x31

Power and Reset

P	set/read power state (from power board)
R	reset/reboot the microcontroller and flash

Help and Manual Control

?	print a help message and enable input echo, for simpler direct human control
/	disable input echo, for returning to program-controlled operation after using command “?”

Advanced and Special Purpose

Test patterns

H	put test patterns in FRAM (disabled if memory is locked)
h	put test patterns in flash (disabled if memory is locked)

Elapsed time and address state manipulation

e	set/read FRAM position/loops (command 'S' modifies this if not armed or recording)
L	set/read current data-recording address in data memory (block and page)

Extra advanced

I	send raw i2c data
i	receive raw i2c data
J	enter manual menus
j	enable/disable debug messages

APPENDIX C – COMMAND DESCRIPTIONS IN DEPTH

The format of each command and response contains zero or more fields, expressed with angle brackets as *<field>*. If a command field is in brackets, like *[<field>]*, it is optional. If multiple fields are in the same set of brackets, like *[<field1> <field2>]*, both must be present or absent together. If a field is in parentheses with an asterisk, like *(<field>)**, there may be zero, one, or more copies of the field, up to a command-specific maximum.

Many commands can both read and write a configuration value. In this case, there will generally be an optional field in the command, with a corresponding field in the response. If the optional field is present, the configuration value is being written, and the response will be *K* if successful; if the optional field is absent, the configuration value is being read, and the response will be the same type as the optional field. A few commands of this type, which read and write multiple values, have multiple optional fields and multiple response fields.

Command descriptions which involve flash will refer to MAXPAGE. This is the maximum valid page address in flash, and is ***one less than*** the response from command =:

MAXPAGE = (response from command =) - 1

The value of MAXPAGE may not be the same across multiple recorder units, as different units may have different numbers of bad blocks in their flash chips; if more blocks are bad, there will be fewer good pages in which to record data, so MAXPAGE will be less. (The actual position of the bad blocks in the chip is invisible to the user; from the user's perspective, bad blocks are nonexistent, and all good pages are at consecutive addresses from 0 to MAXPAGE.)

Commands that require a channel index will use values from the right hand side of the table in the Channel Indexing section. This will lead to a value of 0 referring to analog input 6, for example, which is very confusing if you don't notice that the value is an internal channel index.

Example command description

c <mandatoryField> [<optionalField>]

Command fields:

mandatoryField: A field that must always be present.

optionalField: A field that may or may not be present; its presence or absence will affect what the command does, usually whether the command reads or writes a configuration value.

Response: *<responseField>* (if command argument not given) ***or*** *K*

Response fields:

responseField: A field in the response; will only be present if *optionalField* is absent, as stated in the parentheses above

Whatever the command does will be explained here.

This is a fake command, but it's similar to the many commands of the "set/read" variety. As such, *mandatoryField* would be some sort of address or other selector, and *optionalField* and *responseField* would both be of the same type. In descriptions of this kind of command, *optionalField* and *responseField* will often have the same name, but they don't here.

A paragraph in italics at the end of the description, like this, will be present whenever the

command affects the configuration of the recorder. The paragraph will tell you whether the command's modifications are saved to nonvolatile memory immediately, or whether command X *is required to save the changes. See the Recorder Configuration section for more information.*

Examples:

The entries in the Examples sections do not include an address character. To send a command to a specific unit, put its address character immediately before the command character. For example, to send command *c* to the unit with address "3", the command line should begin with *3c*. To send command *c* to all units, the command line should begin with *0c*.

c 12 abcd ***(write a value)***

K ***(the write was successful)***

c 17 ***(read a value)***

c156 ***(this is the value that was read)***

1

Command fields: none

Response: *<sramData> <checksum>*

Response fields:

*sramData****: 4096-byte hexadecimal value. The entire current contents of the SRAM data buffer.***

*checksum****: 4-byte hexadecimal value. The checksum of*** *sramData* ***using polynomial*** 0x00210801.

Dump the contents of SRAM over the serial link.

The SRAM contains 204 raw records, which are described in the Understanding Recorded Data section. The first 20 bytes are the first record, the next 20 bytes are the next record, and so on. Only the first 4080 bytes contain record data destined for flash, the rest is unused space.

Examples:

1

*2b9603b128440e1…****(long)****…091a72 23ff25b7*

2 [<framPage>]

Command fields:

framPage: 1-byte hexadecimal value. From 0 to 7f. A page address in FRAM.

Response: *<framData> <checksum>*

Response fields:

*framData****: 4096-byte hexadecimal value. The contents of the given page of FRAM.***

*checksum****: 4-byte hexadecimal value. The checksum of*** *framData* ***using polynomial*** 0x00210801.

Dump a page of FRAM over the serial link.

The whole of FRAM contains 26214 records, which are described in the Understanding Recorded Data section. An FRAM page (as dumped by this command) is not an integer multiple of the size of a record, so each page will contain at least one partial record. This means that different FRAM pages will have different data alignments. The intent is that all pages of FRAM should be dumped then reassembled in order, which will correctly restore the records that straddle page boundaries.

The last 8 bytes of FRAM contain housekeeping data instead of record bytes. The layout of this information is described in the Trigger housekeeping in FRAM section.

Examples:

2

*894e43ad22bce…****(long)****…72be24 c0dea233*

3 [<flashPage>]

Command fields:

flashPage: 3-byte hexadecimal value. From 0 to MAXPAGE. A page address in flash.

Response: *<flashData> <checksum>*

Response fields:

*flashData****: 4182-byte hexadecimal value. The contents of the given page of flash, including Reed-Solomon encoding bytes.***

*checksum****: 4-byte hexadecimal value. The checksum of*** *flashData* ***using polynomial*** 0x00210801.

Dump a page of flash over the serial link.

This command will only work if the unit is in USER mode, since the command requires exclusive control of flash. In any other mode, the command will fail. (Failure only happens if the unit is recording data, warming up, or waiting for a trigger, as the unit's default state is USER mode.)

Each page of data consists of 17 globs, each of which is 240 bytes of data (12 records) followed by 6 bytes of Reed-Solomon ECC. The Reed-Solomon bytes are created by the recorder, but the recorder does not process them; responsibility for performing Reed-Solomon decoding is in the hands of the receiving computer.

The format of the records in the response is given in the Understanding Recorded Data section.

Examples:

3

*09576dd23a21c…**(long)**…0a4512 c1809aa3*

4

Command fields: none

***Response:** <configData>*

Response fields:

*configData**: 640-byte hexadecimal value. The contents of the digital board's nonvolatile configuration memory.***

Dump the configuration memory over the serial link.

Configuration memory consists of 5 128-byte pages containing little-endian data structures: consts, configuration, triggerConfig, armState, trigState in that order. Each data structure has its own 128-byte page.

The structure of this command's response is more fully described in the Understanding Configuration Data section.

Examples:

4

*105b98d2ee7d5…**(long)**…326ca3*

A [<armString>]

Command fields:

armString: String, up to 32 bytes long. Intended as an identifier for the current test. If more than 32 characters are given, the extras will be dropped. Can be left blank; the stored string will still be replaced by the new one, i.e. a blank. Any character other than backspace or newline can be included in the string.

***Response:** K*

Response fields: none

Arms the recorder as soon as possible.

The warmup time must still elapse before the system is actually armed and looking for trigger conditions. A value of zero is recorded for the arm delay.

Write access to flash by subsequent commands is immediately blocked, requiring *E* to clear.
This command will fail if write access to flash has been blocked by a previous command.

Examples:

A Primary test 02Nov2016 10:17:33

K

A ***(blank arm string)***

K

B

Command fields: none

Response: *K*

Response fields: none

Clear all data in FRAM.

Unlike the similar flash-clearing command *b*, using this command is not required before taking new data, as FRAM does not need to be erased before it can be written to. However, if data has been recorded, command *E* must be issued to unlock the flash before this command can be used.

Examples:

B

K

b [<flashPage>]

Command fields:

flashPage: Optional 3-byte hexadecimal value; a page address in flash. From 0 to MAXPAGE. If the value entered is larger than MAXPAGE (available from command =), MAXPAGE is the value used.

Response: *K*

Response fields: none

Clear some or all of the data in flash.

If *flashPage* is provided, all blocks will be cleared up to and including the block containing that page. If no argument is provided, the entire flash chip will be cleared. **In almost every case, the right thing to do is to use no arguments and clear all of flash.**

This command will skip bad blocks, like all other flash-accessing commands.

If data has been recorded, command *E* must be issued to unlock the flash before this command, or any other command that writes to flash, can be used.

This command **must** be used to clear any existing data before new data can be safely recorded. After issuing command *E* it is possible to record new data, but the old data must also be cleared, or the new data and the old data will interfere and be unreadable. (This is just the way flash memory works.)

Examples:

b ***(clear all of flash)***

K

b 123cd ***(clear flash from block 0 through block 0x248, which is ceiling(0x123cd/0x80))***

K

C

Command fields: none

Response: *K*

Response fields: none

Manually trigger an armed recorder.

This command only returns successfully if the recorder is armed (i.e. in mode Arm-First or Arm-Again; see the Operating Modes section for an explanation of modes). Otherwise this command will respond with the error "*!*".

Examples:

C

K

D <delaySeconds> [<armString>]

Command fields:

delaySeconds: Decimal value. From 0 to 999999999. The number of seconds to delay arming.

armString: String, up to 32 bytes long. Intended as an identifier for the current test. If more than 32 characters are given, the extras will be dropped. Can be left blank; the stored string will still be replaced by the new one, i.e. a blank. Any character other than backspace or newline can be included in the string.

Response: *K*

Response fields: none

Arms the recorder after a given time delay.

The delay time will be spent in a low-power mode, with the analog signal amplifiers and accelerometers off. After the delay period ends, the warmup time must still elapse before the system is actually armed.

Write access to flash by subsequent commands is immediately blocked, requiring *E* to clear.

This command will fail if write access to flash has been blocked by a previous command.

Examples:

D 86400 Delayed test 03Nov2016 10:17:33 ***(delay for 24 hours)***

K

D 1800 ***(delay for three minutes, blank arm string)***

K

d

Command fields: none

Response: *<secondsRemaining>*

Response fields:

secondsRemaining: Decimal value. The number of seconds until the current delay mode (Delay-Arm or Warmup) is complete.

Gets the amount of time remaining in the current delay mode (Delay-Arm or Warmup). If the unit is not in a delay mode, this command returns zero.

This command is often best preceded by *W*, to determine if the unit is currently in a delay mode. If the unit is not, there's no point in using this command.

Examples:

d

120 ***(two minutes left in current delay mode)***

d

0 ***(less than a second left in current delay mode, or not in a delay mode)***

E

Command fields: none

Response: *K*

Response fields: none

Enable write access to flash memory after data has been recorded.

This is a safety feature, intended to prevent recorded data from being accidentally overwritten and lost. Any time data is recorded (i.e. whenever the system enters an armed state, such as when command *A* is used or the Delay-Arm mode ends), arming and erasing is locked out until this command is issued.

Don't issue this command until you're well and truly done with the contents of the recorder's memory! This command makes that information vulnerable to loss, and there is no easy way to re-enable the lockout.

This command will fail if the recorder is armed, recording, or basically doing anything active or time-dependent. Again, this is for the safety of the data being recorded (or soon to be recorded).

Examples:

E

K

e [<loops> <position>]

Command fields:

loops: 4-byte hexadecimal value. From 0 to 0xfffff. The large-scale current time since arming, expressed in the number of loops that *position* has made through FRAM address space.

position: 4-byte hexadecimal value. From 0 to 0x3ffff. The small-scale current time since arming, expressed in the position in FRAM address space.

Response: *<loops> <position>* (if command fields not given) ***or*** *K*

Response fields:

loops: 4-byte hexadecimal value. The large-scale current time since arming, expressed in the number of loops that *position* has made through FRAM address space.

position: 4-byte hexadecimal value. The small-scale current time since arming, expressed in the position in FRAM address space.

This command tells the user how long it's been since the recorder was armed. See the Time Reporting section for details, but the short version (assuming command *Q* returns the default value of 0x21) is:

time_since_arm = (*loops* * 26214 + floor(*position* / 10)) * 4 microseconds

The time can be changed manually, but doing so while taking data is likely to misalign the data subsequently recorded in FRAM, such that records would not be in the expected places. (Such data would appear crazy at first glance, but coherent data would be recoverable by shifting the

downloaded FRAM data such that the stored records line up with their expected positions.)

Command *S* will reset the values returned by this command if the recorder is not already armed or recording.

Examples:

e

00000010 00038000 ***(recorder has been armed for 1.769444 seconds)***

e

00000000 00000000 ***(recorder has most likely not been armed since power-up)***

F <framPage> [<dumpPages>]

Command fields:

framPage: 1-byte hexadecimal value; a page address in FRAM. From 0 to 7f. The page to start the dump.

dumpPages: Optional 1-byte hexadecimal value. From 1 to 0x80-*framPage*. How many pages to dump. Defaults to 1 if not specified.

Response: *K*

Response fields: none

Dump FRAM data over SERDES.

Each FRAM page is 4096 bytes; 0x80 of these make up the entirety of FRAM. The data is Reed-Solomon encoded and padded before transmission to the controlling PC.

An FRAM page does not contain an integer number of records, so not all FRAM pages will have the same data alignment. The intent is that all of FRAM should be dumped then reassembled via concatenation. The last 8 bytes of FRAM contain housekeeping data instead of record bytes.

Examples:

F 0 80 ***(dump all of FRAM)***

K

F 3 ***(dump page 3 of FRAM)***

K

f

Command fields: none

Response: *K*

Response fields: none

Force-stop all delaying and arming operations, effectively stopping everything that's happening in the background, and switch to USER mode. This allows the user to return the unit to an inactive state without resetting it.

This command fails if the unit is currently recording data (i.e. it has been triggered and is still recording); in this case, the only way to stop the unit is to reset it with command *R*.

Examples:

f

K

G <flashPage> [<dumpPages>]

Command fields:

flashPage: 3-byte hexadecimal value; a page address in flash. From 0 to MAXPAGE. The page to start the dump.

dumpPages: Optional 3-byte hexadecimal value. From 1 to MAXPAGE – *flashPage* + 1. How many pages to dump. Defaults to 1 if not specified.

Response: *K*

Response fields: none

Dump flash data over SERDES.

Each flash page is 4080 bytes prior to encoding. The data is Reed-Solomon encoded before storage, and padded before transmission to the controlling PC.

Examples:

G 0 100 ***(dump the first 256 pages of flash)***

K

G 7 ***(dump page 7 of flash)***

K

g [<dumpPages>]

Command fields:

dumpPages: Optional 3-byte hexadecimal value. How many pages to dump. Valid range depends on how many pages are left in the most recently dumped memory. Defaults to 1 if not specified.

Response: *K*

Response fields: none

Continue the most recent data dump over SERDES. Can continue both FRAM and flash dumps.

If the most recent page dumped was page 4 of FRAM, *g* will dump FRAM starting at page 5. If the most recent page dumped was page 621 of flash, *g* will dump flash starting at page 622.

Examples:

F 20 ***(dump FRAM page 0x20)***

K

g ***(dump FRAM page 0x21)***

K

G 0 40 ***(dump the first 0x40 pages of flash)***

K

g 40 ***(dump the second 0x40 pages of flash)***

K

g 40 ***(dump the third 0x40 pages of flash)***

K

H

Command fields: none

Response: *K*

Response fields: none

Store a test pattern in FRAM. This stores 20 bytes of data at the front of FRAM.

This command fails if data has been recorded and *E* has not been issued.

Examples:

H

K

h

Command fields: none

Response: *K*

Response fields: none

Store a test pattern in flash. This stores one page of data at the front of flash. The test data contains two intentional byte errors in the second 246-byte glob to test the Reed-Solomon correction of the data interpretation software.

This command fails if data has been recorded and *E* has not been issued.

Erasing the flash is not part of this command; if the flash is not erased first, the data written by this command will interfere with the existing data and make a mess of both pages of data.

Examples:

h

K

*I <iicAddress> (<data>)**

Command fields:

iicAddress: 1-byte hexadecimal value. An address on the I2C bus.

data: Up to 64 bytes of hexadecimal data. The bytes are individually parsed, so *00120004*, *00 12 00 04*, and *0 12 0 4* are equivalent.

Response: *K*

Response fields: none

Send raw data over the I2C bus. Best used together with additional documentation!

Examples:

I 13 53 ***(turn off power board output)***

K

i <iicAddress> <dataSize>

Command fields:

iicAddress: 1-byte hexadecimal value. An address on the I2C bus.

dataSize: 1-byte hexadecimal value. How many bytes to receive from the given address. Maximum is 0x40.

Response: *(<byte>)**

Response fields:

byte: 1-byte hexadecimal value. A byte retrieved from the given address. Every byte is followed by a space, even the last one.

Receive raw data over the I2C bus. Best used in conjunction with additional documentation!

Examples:

i 13 1 ***(get power board output state)***

01 ***(power is enabled)***

i 16 2 ***(get power board temperature)***

08 61 ***(25 degrees C)***

J

Command fields: none

Response: (menu is printed)

Response fields: none

Enter the debug menu. This provides low-level access to the data stored in memory, various configuration and status values, and other sorts of things you'd expect in a debug menu.

As the debug menu was intended for use by humans, its input and response formats often differ from those of regular commands. Backspace won't work, but pressing the Escape key will usually clear an entered decimal or hexadecimal value.

This is the only means available of reading the contents of flash blocks in the bad block list.

Examples:

J

c - Analog board tests
p - Power board tests
f - FRAM tests
l - Flash tests
s - SRAM tests
t - triggering tests
T - timer tests
i - i2c tests
m - misc other tests

a - auto self-test (not implemented)

W - watchdog disable

x - Exit

Press a key:

j [<debugEnable>]

Command fields:

debugEnable: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) the printing of debug messages.

Response: *<debugEnable>* (if command argument not given) ***or*** *K*

Response fields:

debugEnable: 1-byte hexadecimal value. Current value of the debug message printing flag.

Enables and disables the printing of debug messages.

Likely to provide more information (and more inscrutable information) than you truly desire, particularly when the recorder is armed or recording data.

Changes to this setting cannot be saved in nonvolatile memory; it is always disabled on reset.

Examples:

j

00 ***(debug messages are disabled)***

j 1 ***(enable debug messages)***

K

K

Command fields: none

Response: *K*

Response fields: none

Repeat the most recent SERDES dump from FRAM. The starting page and number of pages will be the same.

This will repeat dumps from *F*, *g*, or *K* itself. It is intended for cases in which Reed-Solomon error correction or the CRC checksum reveals too many errors in transit, to provide a simple way to retry a failed dump operation.

Examples:

K

K

k

Command fields: none

Response: *K*

Response fields: none

Repeat the most recent SERDES dump from flash. The starting page and number of pages will be the same.

This will repeat dumps from *G*, *g*, or *k* itself. It is intended for cases in which Reed-Solomon error correction or the CRC checksum reveals too many errors in transit, to provide a simple way to retry a failed dump operation.

Examples:

k

K

L [<blockAddress> <pageAddress>]

Command fields:

blockAddress: 2-byte hexadecimal value. From 0 to 7ff. Block address in flash.

pageAddress: 1-byte hexadecimal value. From 0 to 7f. Page address in flash.

Response: *<currentBlock> <currentPage>* (if command arguments not given) ***or*** *K*

Response fields:

currentBlock: 2-byte hexadecimal value. From 0 to 7ff. Current block address in flash.

currentPage: 1-byte hexadecimal value. From 0 to 7f. Current page address in flash.

Reads and writes the block and page addresses in flash where data will be stored next.

Only fully functional while data is not being taken (i.e. not armed or recording; mostly USER mode); in all other circumstances only the block address will be updated. This prevents anything from being stored in the period between the block address change and the page address change.

Useful for determining where the most recent recording session ended and for shifting the location where the next session's data will be stored, for example in a multi-trigger scenario. Since these flash addresses are zeroed when the recorder is armed (specifically, when the

warmup period expires), this command must be issued after that event.

Examples:

L

0012 00

L 400 0 ***(store the next data halfway through flash)***

K

I [<warmupSeconds>]

Command fields:

warmupSeconds: Decimal value. From 1 to 999. Warmup period in seconds.

Response: *<warmupSeconds>* (if command argument not given) ***or*** *K*

Response fields:

warmupSeconds: Decimal value. Current warmup period in seconds.

Reads and writes the number of seconds spent warming up the accelerometers after the recorder is armed.

The recorder will only store data and respond to trigger conditions after this warmup time has elapsed.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

I

120 ***(two minutes of warmup time)***

I 5 ***(warm up for 5 seconds next time the unit is armed)***

K

M [<triggerEnableFlags>]

Command fields:

triggerEnableFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; the channel is enabled as a trigger source if the bit is set, and disabled as a trigger source if not.

For each of the three bytes making up this value, add up the bit values corresponding to the desired channels in the table below to get the value to enter.

Response: *<triggerEnableFlags>* (if command argument not given) ***or*** *K*

Response fields:

triggerEnableFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; the channel is enabled as a trigger source if the bit is set, and disabled as a trigger source if not.

Reads and writes the trigger enable flags for all input channels.

The format for the 3-byte values is given in the table below. Each bit has a corresponding input channel, which is enabled when the bit is 1 and disabled when the bit is 0.

Bit value	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01
1st byte (xx____)	--	--	--	--	trig_ fidu_n	heart_ sync_n	bilevel 6	bilevel 5
2nd byte (__xx__)	bilevel 4	bilevel 3	bilevel 2	bilevel 1	analog 7	analog 8	analog 9	analog 11
3rd byte (____xx)	analog 10	analog 12	analog 1	analog 2	analog 3	analog 5	analog 4	analog 6

Note that each channel's corresponding bit in the 3-byte value is equal to 0x000001 left-shifted by the channel's index. (Channel indices are given in the table in the Channel Indexing section.)

The manual trigger (command *C*) is not in the table because it is always enabled.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X*.*

Examples:

M

0bf001 ***(fiducial input 8 (trig_fidu_n), all bilevel inputs, and analog input 6 are trigger-enabled)***

M 041fc0 ***(enable triggering on fiducial input 7 (heart_sync_n), bilevel input 1, and analog inputs 7 through 12)***

K

m [<lowTriggerFlags> <highTriggerFlags>]

Command fields:

lowTriggerFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; if the bit is set, the channel can cause a trigger when its low threshold is satisfied.

highTriggerFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; if the bit is set, the channel can cause a trigger when its high threshold is satisfied.

For each of the three bytes making up the values, add up the bit values corresponding to the desired channels in the table below to get the value to enter.

Response: *<lowTriggerFlags> <highTriggerFlags>* (if command arguments not given) ***or*** *K*

Response fields:

lowTriggerFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; if the bit is set, the channel can cause a trigger when its low threshold is satisfied.

highTriggerFlags: 3-byte hexadecimal value. Each bit corresponds to a channel; if the bit is set, the channel can cause a trigger when its high threshold is satisfied.

Reads and writes the low and high threshold enable flags for all input channels.

Each channel has a low threshold and a high threshold, which can be enabled independently of each other. A threshold must be enabled and satisfied to cause a trigger event.

- For analog channels, the low and high thresholds are (quiescent value – trigger offset) and (quiescent value + trigger offset); the quiescent value is a running average of recent measurements, and the trigger offset is set with command *O*. If the input value goes *below* the low threshold or *above* the high threshold, the threshold is satisfied.
- For bilevel and fiducial channels, the low threshold is satisfied when the input is low, and the high threshold is satisfied when the input is high. As such, enabling both thresholds for a digital input means the channel's trigger condition will always be satisfied.

The threshold enable flags are independent of the channel's trigger enable flag. However, if the channel's trigger enable flag is disabled, the channel cannot cause a trigger regardless of any thresholds being enabled. (This allows each channel to be enabled and disabled as a trigger source without having to reconfigure which thresholds should be enabled every time.)

The format for the 3-byte values is given in the table below. Each bit has a corresponding input channel, which is threshold-enabled when the bit is 1 and disabled when the bit is 0.

Bit value	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01
1st byte (xx____)	--	--	--	--	trig_ fidu_n	heart_ sync_n	bilevel 6	bilevel 5
2nd byte (__xx__)	bilevel 4	bilevel 3	bilevel 2	bilevel 1	analog 7	analog 8	analog 9	analog 11
3rd byte (____xx)	analog 10	analog 12	analog 1	analog 2	analog 3	analog 5	analog 4	analog 6

Note that each channel's corresponding bit in the 3-byte value is equal to 0x000001 left-shifted by the channel's index. (Channel indices are given in the table in the Channel Indexing section.)

The manual trigger (command *C*) is not in the table because it is always enabled.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X*.*

Examples:

m

0bf001 000231 ***(fiducial input 8 (trig_fidu_n), all bilevel inputs, and analog input 6 have low thresholds enabled, and analog inputs 9, 1, 2, and 6 have high thresholds enabled)***

m 041fc0 7 ***(enable low threshold on fiducial input 7 (heart_sync_n), bilevel input 1, and analog inputs 7 through 12, and enable high threshold on analog inputs 5, 4, and 6)***

K

N [<overlapRecords>]

Command fields:

overlapRecords: 2-byte hexadecimal value. From 0 to 1fff. The minimum number of records of data stored in both FRAM and flash.

Response: *<overlapRecords>* (if command argument not given) ***or*** *K*

Response fields:

overlapRecords: 2-byte hexadecimal value. The current minimum number of records of data stored in both FRAM and flash.

Reads and writes the number of records intentionally stored in both FRAM and flash, to provide observable continuity between the data stored in the two memories.

Because the flash data is buffered in SRAM even before a trigger, on average there will be half a page (102 records) of data stored in both FRAM and flash even if *overlapRecords* is set to zero. However, the size of this innate overlap is determined by the timing of the trigger relative to SRAM wraparound, and may be as low as one record. The value set for *overlapRecords* is in addition to the innate overlap – in other words, *overlapRecords* sets the absolute minimum number of overlapping records between the two memories, but there will probably be more.

The additional overlapping of records provided by *overlapRecords* comes at the expense of data stored in FRAM. The given number of records will be stored in FRAM's circular buffer following the trigger, overwriting the oldest data there. Be sure not to set *overlapRecords* so high that the FRAM doesn't save the amount of pre-trigger data that you require!

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X*.*

Examples:

N

0010 ***(sixteen records minimum of overlap)***

N 5 ***(set minimum overlap to 5 records)***

K

n [<pagesPerTrigger>]

Command fields:

pagesPerTrigger: 3-byte hexadecimal value. From 0 to (MAXPAGE + 1), i.e. the value returned by command =. The number of pages of data recorded in flash per trigger.

Response: *<pagesPerTrigger> <secondsPerTrigger>* (if command argument not given) ***or*** *K*

Response fields:

pagesPerTrigger: 4-byte hexadecimal value. The current number of pages of data recorded in flash per trigger.

secondsPerTrigger: decimal value. The current number of seconds spent recording per trigger, rounded down to the nearest second.

Reads and writes the number of pages of data stored in flash every time the unit is triggered.

The maximum value of *pagesPerTrigger* is the value returned by command *=*.

Each page is 204 records; that's 816 microseconds of recording per page if the between-record delay (command *Q*) is set to the default of 0x21.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X.

Examples:

n

002fdf ***(12255 pages, which is about ten seconds by default, recorded per trigger)***

n 20000 ***(record half the flash per trigger; about 107 seconds by default)***

K

O [<SerDesEnable>]

Command fields:

SerDesEnable: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) SERDES output.

Response: *<SerDesEnable>* (if command argument not given) ***or*** *K*

Response fields:

SerDesEnable: 1-byte hexadecimal value. Current value of the SERDES output enable flag.

Enables and disables the SERDES high-speed serial output.

The SERDES output is automatically enabled when it's used by a command, i.e. when commands *F*, *G*, *g*, *K*, and *k* use it to dump data. It isn't automatically disabled by anything, so after it's used for something it'll just keep pumping out the sync pattern "1111100000" at 150 million bits/second. When SERDES is disabled, the output is a constant "1".

The user is not expected to use this command. Esoteric use cases presumably exist; for example, the user might want to minimize the number of changing signal pins, or the user might want to measure the clock rate of the SERDES signal or the FPGA itself (it's 1/3 the SERDES bit rate).

The status of *SerDesEnable* is not saved; SERDES is always disabled on startup but, as

mentioned above, it's enabled whenever necessary.

Examples:

O

01 ***(SERDES output is enabled)***

O 0 ***(disable SERDES output)***

K

o <channel> [<thresholdOffset>]

Command fields:

channel: 1-byte hexadecimal internal index of an analog channel, from 0 to 0xb (decimal 11). Channel indexes are explained in the Channel Indexing section.

thresholdOffset: 2-byte hexadecimal value. From 0 to fff. (Values up to ffff are accepted, but there's no point.) The offset of the trigger thresholds from the quiescent measured value of the channel.

Response: *<thresholdOffset>* (if command argument not given) ***or*** *K*

Response fields:

thresholdOffset: 2-byte hexadecimal value. The offset of the trigger thresholds from the quiescent measured value of the channel.

Reads or writes the value separation between a channel's quiescent value and the high and low thresholds for that channel. An analog channel's measured value must be above or below the quiescent value by at least this much to cause a trigger.

The *thresholdOffset* value is in units of ADC counts, which is the unit of measure reported by command *S* and stored during data recording.

An analog channel's measured value ranges from 0 to fff, and any computed threshold (quiescent value +/– *thresholdOffset*) exceeding that range is clamped to those limits – no threshold can be less than 0 or greater than fff. Since an input value must *exceed* a threshold to trigger it, a value of *thresholdOffset* greater than or equal to fff means that the channel's thresholds can never be satisfied.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X*.*

Examples:

Z 2 42 ***(set threshold for internal channel index 2 (analog input 5))***

K

Z b ***(read threshold for internal channel index 11 (analog input 7))***

01c0

P [<powerState>]

Command fields:

powerState: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) the output from the power board within the unit.

Response: *<powerState>* (if command argument not given) ***or*** *K*

Response fields:

powerState: 1-byte hexadecimal value. Current state of the power board output.

Enables and disables the power output of the recorder's power board.

If power output is disabled, the recorder is powered off, and will naturally not respond to any further commands.

Examples:

P

01 ***(power is enabled)***

P 0 ***(disable power board output)***

K ***(or potentially no response, as the power was just cut)***

p <powerDataSelect>

Command fields:

powerDataSelect: 1-byte hexadecimal value. Which value to retrieve from the recorder's power board. Can take any of the following values: 1, 2, 4, 5, 6.

Response: *<powerData>*

Response fields:

powerData: 2-byte hexadecimal value. The value retrieved from the power board.

Gets data or voltages from the power board within the unit. The values available are:

powerDataSelect	Value	Interpretation
1	Initialization count	Times power has been applied to unit
2	Power enabled count	Times power has been enabled

4	Capattery voltage	$V_{cap} = \frac{X}{495} = 0.0020203 \cdot X$
5	Battery voltage	$V_{bat} = \frac{X}{495} = 0.0020203 \cdot X$
6	Temperature	$T = 0.020553 \cdot X - 20.41$ (deg. C)

(To sate your curiosity: Option 3 is what command *P* uses. It's different from the others.)

Some sample temperature value conversions:

Value	Temperature
0x0014 (20)	-20°C
0x03E1 (993)	0°C
0x07AE (1966)	20°C
0x08A1 (2209)	25°C
0x16E3 (5859)	100°C

Examples:

p 2 ***(times power has been enabled)***

002b

p 6 ***(temperature)***

07b0 ***(a bit over 20°C)***

Q [<betweenRecordDelay>]

Command fields:

betweenRecordDelay: 2-byte hexadecimal value. From 0 to ffff. The number of processor cycles to pause between the end of one record's acquire-and-store operation and the beginning of the next record's acquire-and-store operation.

Response: *<betweenRecordDelay>* (if command argument not given) ***or*** *K*

Response fields:

betweenRecordDelay: 2-byte hexadecimal value. The number of processor cycles to pause between the end of one record's acquire-and-store operation and the beginning of the next record's acquire-and-store operation.

Reads and writes the length of the pause between record acquisition cycles. This determines the

sampling rate of the recorder.

An explanation of this command's effect on the sampling rate can be found in the Time Reporting section. The short version is that, for a sampling rate of 250,000 samples per second, leave *betweenRecordDelay* set to its default of 0x21. The maximum recording rate, with a *betweenRecordDelay* of zero, should work fine but hasn't been rigorously tested.

The following table contains some example *betweenRecordDelay* values, with their effects:

betweenRecordDelay	Cycles/sample	us/sample	Sampling rate (ksps)
C2A9 (49833)	50000	1000	1
12E1 (4833)	5000	100	10
0341 (833)	1000	20	50
014D (333)	500	10	100
0053 (83)	250	5	200
0021 (33)	200	4	250
0000 (0)	167	3.34	299.4
brd	*brd* + 167	(*brd* + 167) / 50	50000 / (*brd* + 167)

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X.

Examples:

Q

0021 ***(default, 250,000 samples per second)***

Q 0 ***(maximum sampling rate, about 299,400 samples per second)***

K

q <channel> [<triggerMinCount>]

Command fields:

channel: 1-byte hexadecimal internal index of an input channel, from 0 to 0x13 (decimal 19). Channel indexes are explained in the Channel Indexing section.

triggerMinCount: 2-byte hexadecimal value. From 0 to ffff. The minimum number of consecutive threshold-satisfying records needed to cause a trigger on the given channel.

Response: *<triggerMinCount>* (if command argument not given) ***or*** *K*

Response fields:

triggerMinCount: 2-byte hexadecimal value. The minimum number of consecutive threshold-satisfying records needed to cause a trigger on the given channel.

Read or write the minimum number of consecutive threshold-satisfying records needed for a

channel to cause a trigger. Any non-threshold-satisfying record will restart the count from zero.

Setting a channel's *triggerMinCount* to 0 or 1 will cause a trigger during the first record that satisfies an enabled threshold on that channel.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X*.*

Examples:

q c 5 ***(require 5 consecutive threshold-satisfying records to trigger on digital input 1)***

K

q 4 ***(read requirement for analog input 2)***

0012 ***(eighteen consecutive threshold-satisfying records required)***

R

Command fields: none

Response: *K*

Response fields: none

Reset the recorder's microcontroller, FPGA fabric elements, and flash chip.

This resets most settings to their saved values. The settings of the analog boards, which are set by commands *V*, *Y*, *y*, and *Z*, are not reset.

Examples:

R

K

(reset takes effect)

Hello, world!

(etc.)

S

Command fields: none

Response: *<analogVals> <digitalVals> <internalAnalogVals>*

Response fields:

*analogVals****: 12 separate 2-byte hexadecimal fields. The current measurements of the analog inputs, sorted by ascending channel index.***

digitalVals: ***8 separate 1-byte hexadecimal fields. Each field contains the current value of a single bilevel or fiducial input, sorted by ascending channel index. Either 00 or 01.***

internalAnalogVals: ***2 separate 2-byte hexadecimal fields. The current measurements of Vcap and Vbat (capattery and battery voltages) according to the SmartFusion's internal ADCs.***

This table shows how the response fields correspond to channel indexes, analog/bilevel/fiducal inputs, and internal voltages. The top row is the response from the example *S* command.

S resp.	*0b38*	*08a9*	*0b8c*	*094d*	*09b4*	*0b06*	*0ac7*	*0a09*	*0af5*	*0b2a*	*0974*	*0b71*	*00*	*00*	*00*	*00*	*00*	*00*	*01*	*01*	*0450*	*0445*
Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*Vcap*	*Vbat*
Input	*a6*	*a4*	*a5*	*a3*	*a2*	*a1*	*a12*	*a10*	*a11*	*a9*	*a8*	*a7*	*b1*	*b2*	*b3*	*b4*	*b5*	*b6*	*f7*	*f8*	*Vcap*	*Vbat*

Measure the current values of all inputs, as taken directly from the various ADCs and input pins. The analog inputs' values are reported in units of ADC counts.

This command will return wrong analog channel values during Delay-Arm mode, as the analog signal amplifiers and accelerometers are shut down during the delay. **Bilevel channel values may also be incorrect during Delay-Arm;** although the comparators are not shut down, the DACs that generate the threshold voltages (as set by command *Z*) are shut down.

If the unit is already taking data, the values will be skimmed from the running collection process. If not, the unit will be momentarily armed to collect data. (This doesn't affect the contents of FRAM or flash memory, and doesn't lock access to flash the way normal arming does. It does, however, clear the current FRAM address (which is returned by command *e*).)

Examples:

S

0b38 08a9 0b8c 094d 09b4 0b06 0ac7 0a09 0af5 0b2a 0974 0b71 00 00 00 00 00 00 01 01 0450 0445

s

Command fields: none

Response: *<status>*

Response fields:

status: ***A string containing the unit's status. It consists of several sub-fields separated from each other by two spaces, each of which begins with one of the following:***

Subfield Header	Subfield Contents
UNIT LABEL:	Unit's label (as set by command *u*), padded to 32 bytes with \0 chars.
S/N:	Unit's serial number (as set by command *U*).
MODE:	The current operating mode.

SUBMODE:	The current operating submode. Same as command *W*.
STATUS:	Whether a timer is counting down, and if so, its time remaining (as returned by command *d*) and the purpose of the countdown
DATA:	Whether arming is allowed, and if not, whether and when a trigger happened in the stored data
ARM STRING:	The arm string specified as part of the most recent arm command. Same as command *z*.
PAGES RECORDED:	The number of pages of data recorded in the flash. Only checks that the data is present, not whether it's valid. Same as command _. *This subfield is only present when the unit is in USER mode.*
PAGES FUNCTIONAL:	The number of flash memory pages that can hold data (i.e. not part of bad blocks). Same as command =.
BAD BLOCKS:	The number of bad blocks present in the bad block table (and thus, presumably, in flash). Same as the first response field of command -.
UNIT ADDRESS:	The address of the current unit. Expressed as a hexadecimal value (as in command *+*). If the address is a printable character, the hexadecimal value will be followed by the corresponding character, e.g. "*42 (B)*"
VERSION:	***Version ("VER") and compilation date ("COMP") of the unit's firmware***
DONE	(Nothing, "DONE" is the end of the response)

Read the high-level status of the unit in a single string.

For more information about modes, see the Operating Modes section.

The unit label and arm string are padded to 32 bytes with null bytes, which makes printing the full output of this command (as well as *u*, *w*, and *z*) more tricky. Using printf("%s") will end the printing when the nulls start; one alternative is fwrite(), which can make sure that all the characters get printed.

Examples:

s

UNIT LABEL: AER-006 Hangar Queen S/N: 6 MODE: ARM_FIRST SUBMODE: W
STATUS: Warming up, seconds left: 3 DATA: Data present, cannot arm; untriggered
ARM STRING: initial demo PAGES RECORDED: 00000000 PAGES
FUNCTIONAL: 00040000 BAD BLOCKS: 00 UNIT ADDRESS: 48 (H) VERSION:
VER_2014_Jun 14 2016_COMP_Jun 14 2016 17:10:57 DONE

s

UNIT LABEL: AER-006 Hangar Queen S/N: 6 MODE: USER SUBMODE: L STATUS: Timer idle DATA: Data present, cannot arm; trigger occurred at T_arm + 1.983872 seconds ARM STRING: sec delay test PAGES RECORDED: 00000014 PAGES FUNCTIONAL: 00040000 BAD BLOCKS: 00 UNIT ADDRESS: 36 (6) VERSION: VER_2014_Jun 14 2016_COMP_Jun 14 2016 17:10:57 DONE

T [<verboseMode>]

Command fields:

verboseMode: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) verbose output. Defaults to disabled if not given.

Response: *[<verboseStuff>] <resultCode> <testResult>*

Response fields:

verboseStuff: A string containing individual errors as they are discovered. May contain newlines, which complicates parsing when verbose mode is enabled. Only contains information about FRAM errors.

resultCode: 1-byte hexadecimal value. Zero if all tests were successful. Bits that are set correspond to tests that failed or couldn't be run.

testResult: A string containing an explanation of the *resultCode* value.

Runs a self-test of the hardware connections to the FRAM and flash memory chips.

FRAM communication is tested in more depth than flash. When verbose mode is enabled, this command will try to determine which FRAM data/address lines are disconnected. A control line being disconnected could cause many data/address lines to be reported as bad.

Bits set in *resultCode* indicate various errors:

Bit value	Meaning
0x00	No errors
0x01	FRAM test 1 failed
0x02	Flash test failed
0x04	FRAM test 2 failed
0x80	Couldn't run FRAM tests because data is present

Examples:

T

00 All tests OK.

T

80 Unable to test FRAM when data is present.

T 1 ***(verbose output enabled; in this example all possible errors are illustrated)***

Error in data line: 8 9 13 Error in address line: 3 4 Data mismatch at address 00000008
Data mismatch at address 00000010
FRAM test complete.
07 FRAM connection error. Flash communication error.
K

t [<multiTrigEnable>]

Command fields:

multiTrigEnable: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) multiple triggering.

Response: *<multiTrigEnable>* (if command argument not given) ***or*** *K*

Response fields:

multiTrigEnable: 1-byte hexadecimal value. Current value of the multi-trigger enable flag.

Enables and disables multiple triggering.

When multiple triggering (aka multi-trigger) is enabled, after the first set of data is taken, the recorder switches to mode ARM_AGAIN and waits for another trigger condition. (Commands may be used at any time to redefine the conditions for triggering or the amount of data to be stored.)

Each trigger causes the collection of the given number of pages in flash (set with command *n*) until power is lost, flash is full, command *R* is used, or command *f* is used while not recording.

After the first trigger, FRAM is not written to; no other triggers have a fixed amount of pre-trigger data, but because SRAM is used as a circular buffer while waiting for the next trigger, an average of ~100 records of pre-trigger data will be available for each subsequent trigger.

This command does not modify nonvolatile memory. To save any changes made to these settings, use command X.

Examples:

t

00 ***(multi-trigger is disabled)***

t 1 ***(enable multi-trigger)***

K

U [<serialNumber>]

Command fields:

serialNumber: Decimal value. From 0 to 999999999. Unit's serial number.

Response: *<serialNumber>* (if command argument not given) ***or*** *K*

Response fields:

serialNumber: Decimal value. Unit's serial number.

Read or write the serial number of the recorder.

This has no effect on the unit's operation; it's simply for the purpose of telling different units apart. The status command *s* and identity command *w* include this value in their outputs.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

U

16

U 2001

K

u [<unitLabel>]

Command fields:

unitLabel: String, up to 32 bytes long. Intended for unique identification of the unit.

Response: *<unitLabel>* (if command argument not given) ***or*** *K*

Response fields:

unitLabel: String, up to 32 bytes long.

Read or write the description string of the recorder.

This has no effect on the unit's operation; it's simply for the purpose of telling different units apart. The status command *s* and identity command *w* include this value in their outputs.

The string is padded to 32 bytes with null bytes, which makes printing the full output of this command (as well as *s*, *w*, and *z*) more tricky. Using printf("%s") will end the printing when the nulls start; one alternative is fwrite(), which can make sure that all the characters get printed. Since nothing follows the nulls on the response line, it's actually okay to just use printf() for this command, but not for commands such as *s*.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

u

AER-006 Hangar Queen

u Production Unit 15

K

V <boardSelect> <referenceSelect> [<voltageValue>]

Command fields:

boardSelect: 1-byte hexadecimal value. From 1 to 3. Each board handles 4 analog inputs and 2 bilevel inputs.

Board	Analog inputs	Bilevel inputs
1	1-4	1, 2
2	5-8	3, 4
3	9-12	5, 6

referenceSelect: 1-byte hexadecimal value. Either zero or nonzero. If zero, refers to the 2.000V reference (internal bias); if nonzero, refers to the 2.222V reference (output bias).

voltageValue: 2-byte hexadecimal value. A value used by a DAC to generate the reference voltage selected by *referenceSelect*. Ideally, the resulting voltage is (*voltageValue*/0x10000)*5V.

Response: *<voltageValue>* (if command argument not given) ***or*** *K*

Response fields:

voltageValue: 2-byte hexadecimal value. The value currently used by a DAC to generate the reference voltage selected by *referenceSelect*. Ideally, the resulting voltage is (*voltageValue*/0x10000)*5V.

Read or write the value being used by an analog board to generate a reference/bias voltage.

See the analog board documentation for more details, but the short version is as follows: The 2.000V reference voltage provides a center point for the post-offset input, and the 2.222V reference voltage drives the final output filter.

This command does not modify nonvolatile memory. This setting will be reset to the saved value upon power cycle, but not when command R *is issued. To save a new value or revert to the saved value, use command* X*.*

Examples:

V 2 0

6666 ***(default 2 volt value)***

V 3 1 71c7 ***(default 2.222 volt value)***

K

v [<armOnPower-upFlag> [<delaySeconds> <armString>]]

Command fields:

armOnPower-upFlag: 1-byte hexadecimal value. Either zero (do not arm on power-up) or nonzero (arm on power-up). This value can be set without altering the other two stored values by including only this field in the command.

delaySeconds: Decimal value. From 0 to 999999999. The number of seconds to delay arming.

armString: String, up to 32 bytes long. Intended as an identifier for the current test. If more than 32 characters are given, the extras will be dropped. Can be left blank; the stored string will still be replaced by the new one, i.e. a blank. Any character other than backspace or newline can be included in the string.

Response: *<armOnPower-upFlag> <delaySeconds> <armString>* (if no command arguments are given) *or !* (if flash write access is blocked) ***or*** *K*

Response fields:

armOnPower-upFlag: 1-byte hexadecimal value. Either 00 (do not arm on power-up) or 01 (arm on power-up).

delaySeconds: Decimal value. The number of seconds to delay arming.

armString: The string to be recorded as the arming string when the arm-on-power-up event actually happens.

Reads and writes the recorder's arm-on-power-up configuration. These settings allow the recorder to automatically arm itself at some time after its power supply is enabled. The effect is equivalent to a *D* command applied immediately upon power-up, using the *delaySeconds* and *armString* values stored by this command.

When a power-up or reset happens, this is what occurs:

1. If arm-on-power-up is disabled, do nothing special; ignore the rest of this list
2. Print a message describing what's about to happen (failure to arm, immediate arm, or delayed arm)
3. If flash write access is blocked, the arming attempt failed; ignore the rest of this list
4. If this command's *delaySeconds* is zero, arm immediately as in command *A* using the *armString* value from this command, and disable arm-on-power-up to prevent a second automatic arming; ignore the rest of this list
5. Enter Delay-Arm mode as in command *D* using the *delaySeconds* and *armString* values from this command
6. If Delay-Arm mode finishes without interruption, disable arm-on-power-up to prevent a second automatic arming, and block write access to flash (requires *E* to clear)

7. If Delay-Arm mode is interrupted (e.g. with commands *f*, *A*, or *D*, or a loss of power), leave arm-on-power-up enabled for next time

The write action of this command (i.e. using the command with any command arguments at all) will fail if write access to flash has been blocked by a previous command. This prevents the user from setting up an arm-on-power-up that cannot be carried out.

It's possible to enable/disable arm-on-power-up without changing the stored *delaySeconds* and *armString* values by only including *armOnPower-upFlag* in the command.

An arm-on-power-up recording session will probably not include adjusting the configuration between power-up and arming, so be sure that the configuration saved in nonvolatile memory with command *X* is satisfactory before attempting arm-on-power-up recording.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

v ***(read status)***

01 180 This is an arm string ***(arm-on-power-up enabled, delay for three minutes after power-up)***

v 0 ***(disable arm-on-power-up, leave other settings alone)***

K

v 1 ***(enable arm-on-power-up using existing settings)***

K

v 1 86400 Test01 05Nov2016 07:12:56 ***(enable arm-on-power-up, delay for 24 hours after power-up)***

K

W

Command fields: none

Response: *<currentMode>*

Response fields:

*currentMode****: A single character reflecting the current state of the recorder.***

Read the current mode of the unit – whether it's armed, recording data, has recorded data, and so on. See the "Submode" column of the table in the Operating Modes section to interpret this command's response.

Examples:

W

U ***(User mode)***

W

F ***(Arm-First mode)***

w

Command fields: none

Response: *<identity>*

Response fields:

*identity****:*** A string containing the unit's essential identity. It consists of several sub-fields separated from each other by two spaces, each of which begins with one of the following:

Subfield Header	*Subfield Contents*
Unit label:	***Unit's label (as set by command*** *u****), padded to 32 bytes*** with \0 chars
S/N:	***Unit's serial number (as set by command*** *U****)***
Version:	***Version ("VER") and compilation date ("COMP") of the unit's firmware***

Read the identity of the unit in a single string.

The unit label is padded to 32 bytes with null bytes, which makes printing the full output of this command (as well as *s* and *u*) more tricky. Using printf("%s") will end the printing when the nulls start; one alternative is fwrite(), which can make sure that all the characters get printed.

Examples:

w

Unit label: AER-006 Hangar Queen S/N: 6 Version: VER_2005_Dec 21 2015_COMP_Dec 21 2015 20:47:23

X <saveConfiguration>

Command fields:

saveConfiguration: 1-byte hexadecimal value. Either 0 (revert configuration from saved values) or nonzero (save current configuration to nonvolatile memory).

Response: *K*

Response fields: none

Save or revert the current configuration of the unit to the values stored in nonvolatile memory.

Almost all commands that read or write a setting only read or write the temporary, stored-in-RAM version of that setting; to save such a setting's value for future use, this command must be used. The temporary settings, not the saved settings, are the ones in effect, but they will reset to the saved versions on the next power-up or, for most settings, the next reset (command *R*).

All settings that can be saved and reverted will be saved or reverted by this command. This includes the settings stored on the analog boards.

Because the saved values are loaded on power-up, they will most likely be the values used when doing arm-on-power-up recording, so be sure to test that the values loaded on power-up are the values you want to use.

Examples:

X 1 ***(Save current configuration)***

K

X 0 ***(Revert to saved configuration)***

K

Y <channel> [<offset>]

Command fields:

channel: 1-byte hexadecimal internal index of an analog channel, from 0 to 0xb (decimal 11). Channel indexes are explained in the Channel Indexing section.

offset: 2-byte hexadecimal value. The offset added to the analog input voltage prior to the application of gain. Ideally, the offset voltage is (*offset*/0x10000)*5V.

Response: *<offset>* (if command argument not given) ***or*** *K*

Response fields:

offset: 2-byte hexadecimal value. The offset added to the analog input voltage prior to the application of gain. Ideally, the offset voltage is (*offset*/0x10000)*5V.

Read or write the offset value for analog inputs. The offset value set with this command is passed directly to a DAC on the proper analog board.

The intent is that the quiescent value of each analog input will be offset (using this command) to be equal to the 2.000V reference voltage for a particular analog board. This will cause any deviation from that quiescent value to be amplified by a gain factor (set by command *y*). The reference voltage is set using command *V*.

This command does not modify nonvolatile memory. This setting will be reset to the saved value upon power cycle, but not when command R *is issued. To save a new value or revert to the*

saved value, use command X.

Examples:

Y 0 ***(read offset of analog input 6)***

6666

Y 7 4099 ***(set offset of analog input 10)***

K

y <channel> [<gain>]

Command fields:

channel: 1-byte hexadecimal internal index of an analog channel, from 0 to 0xb (decimal 11). Channel indexes are explained in the Channel Indexing section.

gain: 1-byte hexadecimal value. The gain factor applied to the input voltage after offset but before filtration and measurement. Can be any of the following values: 1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80.

Response: *<offset>* (if command argument not given) ***or*** *K*

Response fields:

gain: 1-byte hexadecimal value. The gain factor applied to the input voltage after offset but before filtration and measurement.

Read or write the gain factor for analog inputs.

The gain factor value set with this command is applied after the offset value set with command *Y*. This gain is applied to the difference between the post-offset channel input voltage and the analog board's 2.000V reference voltage; in other words, if the post-offset input voltage is the same as the reference voltage, a higher gain won't affect the output.

This command does not modify nonvolatile memory. This setting will be reset to the saved value upon power cycle, but not when command R *is issued. To save a new value or revert to the saved value, use command* X.

Examples:

y 0 ***(read gain of analog input 6)***

01

y 7 20 ***(set gain of analog input 10 to 0x20)***

K

Z <channel> [<threshold>]

Command fields:

channel: 1-byte hexadecimal index of a digital input, from c (decimal 12) to 11 (decimal 17). Channel indexes are explained in the Channel Indexing section.

threshold: 2-byte hexadecimal value. A value used by a DAC to generate the threshold voltage. Ideally, the resulting voltage is (*threshold* /0x10000)*5V. Maximum value is 0xa8f5 (decimal 43253), for a maximum voltage of 3.3V.

Response: *<threshold>* (if command argument not given) ***or*** *K*

Response fields:

threshold: Current threshold value of the selected digital input. Ideally, the threshold voltage is (*threshold* /0x10000)*5V.

Read or write the threshold voltage of a digital input. The digital channel value will be high if and only if the voltage on the channel's input line exceeds the threshold voltage.

The maximum allowed threshold voltage is 3.3 volts. This is because the voltage runs to a comparator powered by a 3.3V supply; the comparator could be damaged if its inputs much exceed 3.3V. (Each bilevel input signal is clamped by hardware to a range of 0V to 3.3V, so the input signals won't damage the comparators either.)

This command does not modify nonvolatile memory. This setting will be reset to the saved value upon power cycle, but not when command R *is issued. To update the saved value or revert to the saved value, use command* X*.*

Examples:

Z c 6789 ***(set threshold for digital input 1)***

K

Z 11 ***(read threshold for digital input 6)***

5999

z

Command fields: none

Response: *<armString>*

Response fields:

armString: String, up to 32 bytes long.

Read the string set by the most recent successful arm operation. This can be used to identify what the most recent test was all about.

The string is padded to 32 bytes with null bytes, which can make printing the full output of this command (as well as *s*, *u*, and *w*) more tricky. Using printf("%s") will end the printing when the nulls start; one alternative is fwrite(), which can make sure that all the characters get printed. Since nothing follows the nulls on the response line, it's actually okay to just use printf() for this command, but not for commands such as *s*.

Examples:

z

Primary test 02Nov2016 10:17:33

; [<Power-downEnable>]

Command fields:

Power-downEnable: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other value) automatic power-down when recording is complete.

Response: *<Power-downEnable>* (if command argument not given) ***or*** *K*

Response fields:

Power-downEnable: 1-byte hexadecimal value. Current value of the flag controlling automatic power down.

Enables and disables automatic power-down.

If this flag is enabled, then the recorder will turn itself off when it reaches the Power-down mode. This happens when the Recording mode records the pre-defined number of records (set by command *n*) with multi-trigger off (set by command *t*), or when the recorder completely fills the flash with recorded data.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

;

00 ***(automatic power-down is disabled)***

; 1 ***(enable automatic power-down)***

K

~ [<heartbeatEnable>]

Command fields:

heartbeatEnable: 1-byte hexadecimal value. Whether to disable (if 0) or enable (any other

value) the heartbeat signal output.

Response: *<heartbeatEnable>* (if command argument not given) ***or*** *K*

Response fields:

heartbeatEnable: 1-byte hexadecimal value. Current value of the heartbeat signal output enable flag.

Enables and disables the heartbeat signal output.

When enabled, the recorder generates a 1Hz square wave heartbeat signal. It appears on the Heart_Sync_n line, which is recorded as part of each record. Disabling the heartbeat signal allows the Heart_Sync_n line to be driven by an external source, providing a fiducial signal which is independent of trigger events. This is particularly useful for a multi-unit system, in which a shared external fiducial provides synchronization of the recorded data on different units.

In a pinch, with the heartbeat signal disabled, Heart_Sync_n could be used as an extra digital input.

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

~

00 ***(heartbeat output is disabled)***

~ 1 ***(enable heartbeat output)***

K

+ [<unitAddress>]

Command fields:

unitAddress: 1-byte hexadecimal value. The ASCII representation of the character to be used for the unit's address. For decimal digits, just add 0x30 to the digit. A table of ASCII characters is available at www.asciitable.com and many other places.

Response: *<unitAddress>* (if command argument not given) ***or*** *K*

Response fields:

unitAddress: 1-byte hexadecimal value. The ASCII representation of the character to be used for the unit's address.

Sets the recorder's address for serial communication.

The value of *unitAddress* is an ASCII value. As such, an argument of *2* will set the address character to an unprintable character. If you want to use the character "2" as the recorder's address, use the ASCII representation of the character "2", which is 0x32; the argument to use for an address character of "2" is thus *32*.

The default value of *unitAddress* is 0xff. This must be changed before the recorder can be used

in a multi-unit system, so that each unit can be individually communicated with. To use this command on an unconfigured recorder, connect it to a serial interface with no other recorders connected, then use the broadcast address "0" to send this command to all connected recorders; since only one recorder is connected to the interface, only its *unitAddress* will be updated. For example, to set a new recorder's address to "1", the complete command line to send would be "*0+ 31*".

Setting *unitAddress* to the value 0x00 will cause the recorder to always act as if its address character has already been received, thus treating the first character on each received line as a command character. Getting the unit out of this state therefore requires a command line such as "*+ 31*", with no initial address character.

A few *unitAddress* values are invalid, and will cause this command to fail. These values are 0x30 (which is the broadcast address character "0") and 0x08 (which is the backspace character).

Changes to this setting are immediately saved in nonvolatile memory.

Examples:

+

34 ***(address character is '4')***

+ 41 ***(set address character to 'A')***

K

=

Command fields: none

Response: *<goodPages>*

Response fields:

goodPages: 4-byte hexadecimal value. The number of functional pages in the flash chip.

This command returns the maximum data capacity of a recorder.

This can vary from unit to unit, as some recorders may have flash chips with bad blocks. Any pages in a bad block cannot be safely used to store data, which reduces the amount of space available for recorded data. This command interprets the bad block list to determine how many pages are left.

The value of *goodPages* is the maximum argument to command *n*, which sets the number of pages to be recorded after a trigger.

Examples:

=

00040000 ***(no bad blocks present)***

=

0003fe80 ***(3 bad blocks present)***

_ (underscore)

Command fields: none

Response: *<firstEmptyPage>*

Response fields:

firstEmptyPage: 4-byte hexadecimal value. The address of the first empty page in flash.

This command returns the amount of data currently recorded in flash.

This command will only work if the unit is in USER mode, since the command requires exclusive control of flash. In any other mode, the command will fail. (Failure only happens if the unit is recording data, warming up, or waiting for a trigger, as the unit's default state is USER mode.)

The first empty page, by definition, is the address following the last recorded page. Since page addressing starts at zero, the response from this command is identical to the number of pages that have been recorded.

Any non-blank pages qualify as "recorded data", so if data is recorded over existing data, the value returned by this command will be the greater of the two recording lengths. (The overlapping parts of the data will be trashed, of course.)

If command *b* was used to clear only some of flash, it's possible that this command will be misled by uncleared data, as this command uses a binary search to find the first empty page. This is easily prevented by always clearing all of flash when using command *b*.

Examples:

_

00000400 ***(1024 pages of data have been recorded in flash)***

_

00040000 ***(262144 pages recorded – no bad blocks present, and all pages contain data)***

_

00000000 ***(no data has been recorded, flash is blank)***

- (hyphen)

Command fields: none

Response: *<badBlocks> (<blockIndex>)**

Response fields:

badBlocks: 1-byte hexadecimal value. The number of *blockIndex* fields in the response.

blockIndex: 2-byte hexadecimal value. The index of a bad block in the flash chip. The bad blocks are listed in ascending order.

This command reveals which blocks in flash memory have been marked as bad.

Under almost no circumstances will the user need to care about this information. However, it is possible that blocks have been marked bad incorrectly; if this command returns sequences of blocks, this may be the case. Incorrect bad-block marking is almost definitely present if block 0 is marked bad, as the manufacturer of the flash chip guarantees that block to be good.

Even if blocks have been incorrectly been marked bad, the only effect on recorder operation is that slightly less space will be available in flash for recorded data.

Examples:

-

00 ***(no bad blocks present)***

-

05 0013 001f 005e 0080 01a9 ***(5 bad blocks present)***

?

Command fields: none

Response: (command list is printed)

Response fields: none

This command prints a list of commands and enables input echo. The intent of this command is to make life easier for a human operator.

"Input echo" means that the unit will send back every character it receives, including the effects of backspace. This makes the "local echo" setting on a terminal program unnecessary, as entered commands will be visible to the user. However, broadcast commands will only be echoed if the computer is connected directly to the recorder, without a serial multiplexer being involved (see the Communicating with Recorders section for details); in this case, local echo may be preferable.

The commands are listed with a brief description, the argument fields they take (*[ARGS]*), and the response fields they return (*[RESP]*).

Examples:

?

1 - serial-read SRAM in ascii with checksum [ARGS] x [RESP] <4096 bytes of data> <4 byte checksum>

2 - serial-read FRAM in ascii with checksum [ARGS] <1 byte page address (0 to 0x7f)> [RESP] <4096 bytes of data> <4 byte checksum>

…

? - print this help message and enable input echo

/ - disable input echo, for GUI-compatible operation

Press a key:

/

Command fields: none

Response: *Disabling input echo, for return to automated use.*

Response fields: none

This command disables input echo, in which the recorder repeats back each character it receives.

Input echo makes a human user's life easier by letting them see what they're typing, but a computer program will probably not expect to receive a copy of the command it just sent. To prevent such response-parsing problems, this command should be sent when switching from human-controlled interaction to computer-controlled interaction.

One option is to have a controlling program send this command when it starts, to ensure that the recorder is in the proper mode for automated control.

Examples:

/

Disabling input echo, for return to automated use.

DISTRIBUTION

AFRL / RWMFI
Attn: Kenneth J. Williamson
306 W. Eglin Blvd, Bldg. 432
Eglin AFB, FL 32542-5430

1	MS0382	Christopher Eiting	2158
5	MS0386	Matthew Brewer	2159
1	MS0661	Anne L Benz	2627
1	MS0661	Jared Bare	2627
1	MS0661	Shane K Curtis	2627
1	MS0661	Roberto A Jimenez	2627
1	MS0661	Michael E Partridge	2627
1	MS1135	James A Dykes	1531
1	MS1135	John C Griffin	1535
1	MS1160	Doug Dederman	5421
1	MS1160	Jason W Brown	5421
1	MS1160	Erik Nishida	5421
1	MS1248	David P McGrogan	5631
1	MS9102	Kathryn R. Hughes	8133
1	MS0899	Technical Library	9536 (electronic copy)


Sandia National Laboratories