	Document # **LAT-TD-331-0**	Date Effective 10 August 2001
	Prepared by(s) J.J.Russell	Supersedes None
GLAST LAT TECHNICAL DOCUMENT	Subsystem/Office Electronics Subsystem	
Document Title **LAT Flight Software Preliminary Design Report**		

Gamma-ray Large Area Space Telescope (GLAST)

Large Area Telescope (LAT)

LAT Flight Software Preliminary Design Report

CHANGE HISTORY LOG

Revision	Effective Date	Description of Changes
0	10 Aug 2001	Original

Table Of Contents

Figures

Tables

1 Glossary and Acronyms

1553	MIL-STD-1553 Bus (Interface between Spacecraft and LAT)
ACD	Anti-Coincidence Detector
ADC	Analog to Digital Converter
AEM	ACD Electronics Module
CAL	Calorimeter
CFG	Configuration
CCSDS Packet	A Spacecraft Packet Telecommand Standard approved by The Consultative Committee for Space Data Systems
DAC	Digital to Analog Converter
DAQ	Data Acquisition System
EM1	Engineering Model 1
EM2	Engineering Model 2
EP	Event Processor
EPU	Event Processing Unit
GAFE	ACD Front-end Electronics module
GARC	ACD Read-out Controller
GBM	Gamma-ray Burst Monitor
GCFE	Calorimeter Front-end Electronics module
GCRC	Calorimeter Read-out Controller
GLT	Global Trigger
GPS	Global Positioning System
GSE	Ground Support Equipment
GTFE	Tracker Front-end Electronics module
GTRC	Tracker Read-out Controller
HI discriminator	A discriminator set for the high end of a range
HSK	Housekeeping
LAT	Large Area Telescope
LO discriminator	A discriminator set for the low end of a range
LRP	Low Rate Physics
SEU	Single Event Upset
SIU	Spacecraft Interface Unit
SSR	Solid State Recorder
TEM	Tower Electronics Modules
TDRSS	Tracking and Data Relay Satellite System
TKR	Silicon Tracker

Table 1 Instrument Terms

AGN	Active Galactic Nuclei
CNO	A heavy nuclei event, consisting of either Carbon, Nitrogen, Oxygen or higher Z
GRB	Gamma Ray Burst
MIP	Minimum Ionizing Particle
SAA	South Atlantic Anomaly

Table 2 Scientific Terms

CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSCI	Computer Software Configuration Item
DMA	Direct Memory Access
EEPROM	Electronically Erasable Programmable Read Only Memory
FQT	Formal Qualification Test
FSW	Flight Software
I/O	Input / Output
kByte	2^{10} bytes
kHz	1000 Hertz
Mbit	2^{20} bits
Mbyte	2^{20} bytes
MMU	Memory Management Unit
OS	Operating System
QA	Quality Assurance
RAM	Random Access Memory
ROM	Read Only Memory
RTOS	Real Time Operating System
SDP	Software Development Plan
VxWorks	Wind River's RTOS

Table 3 Computer Terms

CDR	Conceptual Design Report
PDR	Preliminary Design Report
TBD	To Be Determined
WBS	Work Breakdown Schedule

Table 4 Miscellaneous Terms

2 Overview

To meet the LAT mission requirements, Flight Software (FSW) must, at the highest level, address the following mission objectives:

FSW Mission Objectives	Description
I	Collect and Record LAT Data
II	Transient Event Notification
III	Operational Support Functions
IV	Calibration and Monitoring Activities

Table 5 Flight Software Objectives

I. Collecting, recording and downlinking LAT instrument data is the primary function of flight software. The data collected fall into three major categories:
   - Physics Data.
   - Housekeeping Data.
   - Low Rate Physics Data.

II. Flight Software is also tasked to provide prompt ground notification of transient events such as Gamma Ray Bursts (GRB) and Active Galactic Nuclei (AGN). Such notification can be used bring other astronomical resources to bear on these transitory phenomena.

III. Flight software must also provide a fully functional operational environment. This includes command, control and configuration abilities which in turn rely on reliable communications with the spacecraft.

IV. Through calibration and monitoring activities, flight software endeavours to keep the LAT operating at peak efficiency. Where flight software is unable to correct an anomaly, flight software is responsible for reporting the situation to the ground, either through the regularly scheduled downlinks or, if the anomaly is sufficiently severe, through the TDRSS real time access system.

This section provides an overview of the software architecture needed to achieve the primary goals and provide a foundation for the necessary support services. Subsequent sections describe the major software components of the system in a semi-chronological order, from the boot phase to the output stage.

2.1 Requirements and Documents

The LAT Flight Software and the Electronics Groups work closely in order to achieve an optimal LAT Data Acquisition (DAQ) system. Both groups work to the requirements set out in *Dataflow Level IV Subsystem Specification,* LAT-SS-00285. The complete list of

documents relevant to the LAT Flight Software is:

- LAT-SS-00019, GLAST LAT – T&DF Level III Subsystem Specification
- LAT-SS-00284, GLAST LAT – Trigger Level IV Subsystem Specification
- LAT-SS-00285, GLAST LAT - Dataflow Level IV Subsystem Specification
- LAT-SS-00286, GLAST LAT – Conceptual Design of the Global Trigger
- LAT-SS-00287, GLAST LAT – Conceptual Design of the Electronics, Trigger and Dataflow System
- LAT-SS-00288, GLAST LAT – Conceptual Design of the TKR-CAL Tower Electronics Module
- LAT-SS-00289, GLAST LAT – Conceptual Design of the ACD Electronics Module
- LAT-SS-00290, GLAST LAT – Conceptual Design of the Space-Craft Interface Unit
- LAT-MD-104.1, LAT Flight Software Management Plan

2.2 Software Architecture

The following depicts an idealized view of the FSW architecture:

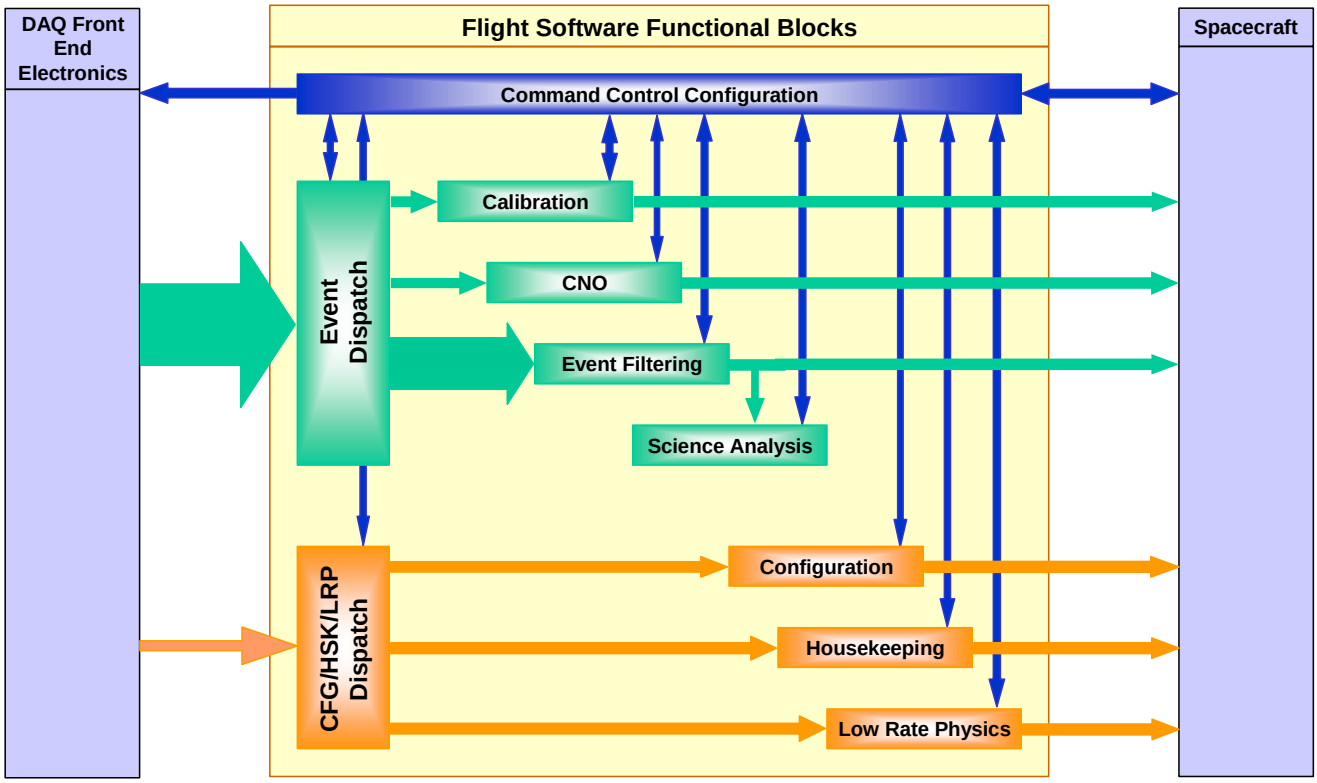

Figure 1 Flight Software Architecture

The front-end electronics provide three major categories of data on each of two input pipes. The low bandwidth (~20 Mbit/sec) pipe delivers non physics event data to the CFG/HSK/LRP dispatcher while the high bandwidth pipe (160 Mbit/sec) pipe delivers physics events to the event dispatcher. Each dispatcher in turn directs the data to the appropriate processing unit.

Each processing unit performs function-specific calculations on the data. The processed data, or some summary of it, is output to the spacecraft through one of two interfaces, either a MIL-STD-1553 Bus, hereafter referred to as the 1553 bus, or a dedicated high-speed serial link (32Mbit/sec) to the on-board Solid State Recorder (SSR).

2.2.1 Configuration / Housekeeping / Low Rate Physics Dispatcher

Three categories of data are transported on this path:

- Configuration Data.
- Housekeeping Data.
- Low Rate Physics Data.

The *CFG/HSK/LRP Dispatcher* classifies incoming data into one of these three categories and directs the data to the appropriate processing unit. A key attribute about data arriving on this pipe is that it is all solicited. A command is sent requesting the desired data and the front-end electronics complies. For simplicity and reliability, the interface supports only a synchronous protocol.

2.2.2 Event Dispatcher

Data on the *Event Dispatcher* input pipe is sourced by the hardware trigger and arrives asynchronously and at high rates. It is a complex path, not only because of its high-speed and asynchronous nature, but also because the hardware must rendezvous data from all sources (16 TEMs + AEM + Global Trigger) before delivering it to the destination.

Three types of data are carried on the event pipe:

- Physics data.
- CNO data.
- Calibration Data.

The first two types of data will be simultaneously present on the data pipe in the regular data acquisition mode. Calibration data, at least in this context[1], arrives only when the detector is run in a special mode. This mode is incompatible with normal running and so calibration data will never be intermingled with physics and CNO data.

Note that data presented to the Event Dispatcher is always in the form of hardware assembled events regardless of how the events were triggered. In this sense, CNO events

[1] Data used to calibrate the detector is also present during normal physics running. This includes CNO events. For purposes of this document, such data will be called monitor data to distinguish it from data taken during dedicated calibration running.

may be regarded as a particular instantiation of a more general class of events triggered solely for the purpose of monitoring the LAT.

2.2.3 Command, Control and Configuration

The data processing units, i.e. the *Calibration, CNO, Event Filtering*, etc., all share the services of the *Command, Control and Configuration* facilities.

Commanding activities occur at two different levels. A low level commanding facility is responsible for directly interacting with the front-end electronics. These low level commands are the building blocks of a high level commanding facility. High level commands are received from the spacecraft and are translated into the low level commands that affect the front-end electronics. For example, the start of data taking will involve three high level commands that:

- Select and load a configuration for the front-end electronics.
- Check that configuration is correctly loaded by reading it back.
- Start data taking by enabling the triggers.

These three high level commands will spawn many individual low level commands.

The high level commanding facility is also responsible for orchestrating the behavior of the LAT. This facility involves not only translating the high level commands to low-level commands, but also scheduling and recording these actions.

The configuration facility is responsible for managing the various front-end configurations that will be used during operations.

2.2.4 Output

The flight software supports two physical output streams that carry a number of logical output streams. One physical stream is responsible for transporting the bulk data, such as the physics, housekeeping and LRP data to the Solid State Recorder, where it is stored until transmitted to the ground. The 1553 bus provides the other physical stream and allows access to the real-time telemetry downlink and the spacecraft.

2.3 A Practical Realization of the Flight Software

The architecture presented in the previous section is independent of the actual hardware platform. This section illustrates a practical realization of this architecture on the anticipated LAT DAQ hardware.

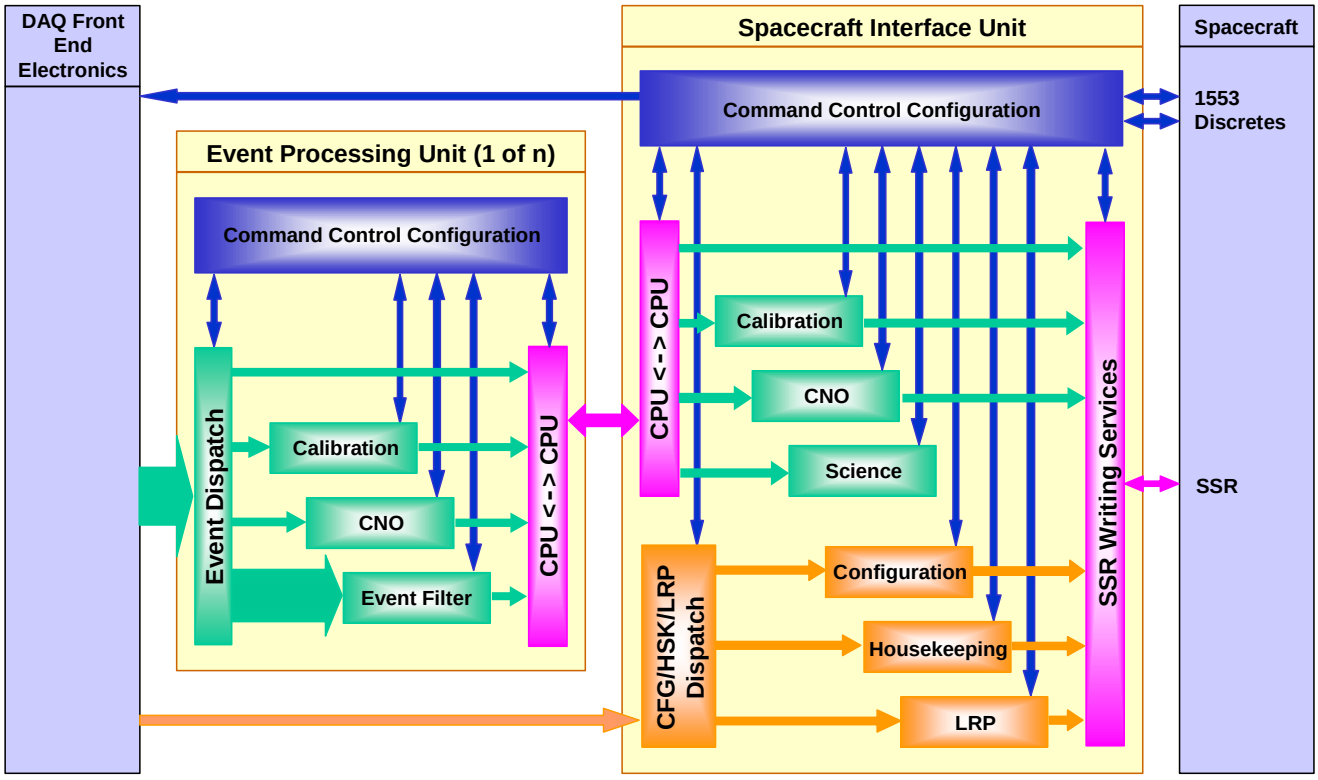

Figure 2 Realization of Flight Software Architecture

The LAT DAQ will consist of a number of Event Processing Units (EPU) and one Spacecraft Interface Unit (SIU). At least one cold spare will be provided for both classes. The exact number of EPUs has not yet been determined, as the number of CPU cycles is driven by the event processing needs. However, initial estimates indicate that approximately four Power PCs 750 @ 150 MHz will be sufficient. The dataflow hardware architecture can accommodate between 1 and 8 EPUs, but for definiteness, this discussion will assume one SIU plus one cold spare and four EPUs plus one cold spare.

The configuration is strictly a master/slave relation between the SIU and the EPUs. The SIU supports all activities except for the event processing. The event processing is entirely the responsibility of the EPUs.

Only the SIU has interconnects with the spacecraft. These interconnects include:

- A 32Mbit/sec link to the SSR.
- A MIL-STD-1553 bus.
- A small number of discretes, including the GRB prompt alert from the GBM.

Contributions from the EPUs must be gathered in the SIU, which is responsible for maintaining statistics on all the data and transporting the data to the Solid State Recorder. The SIU must also synchronize and coordinate the activities of the EPUs. Dedicated point-to-point links between the SIU and the EPUs provide the communication path for these activities.

2.4 The DAQ Platform

For reference, a block diagram illustrating all the major components of the LAT dataflow system and the interconnections is provided:

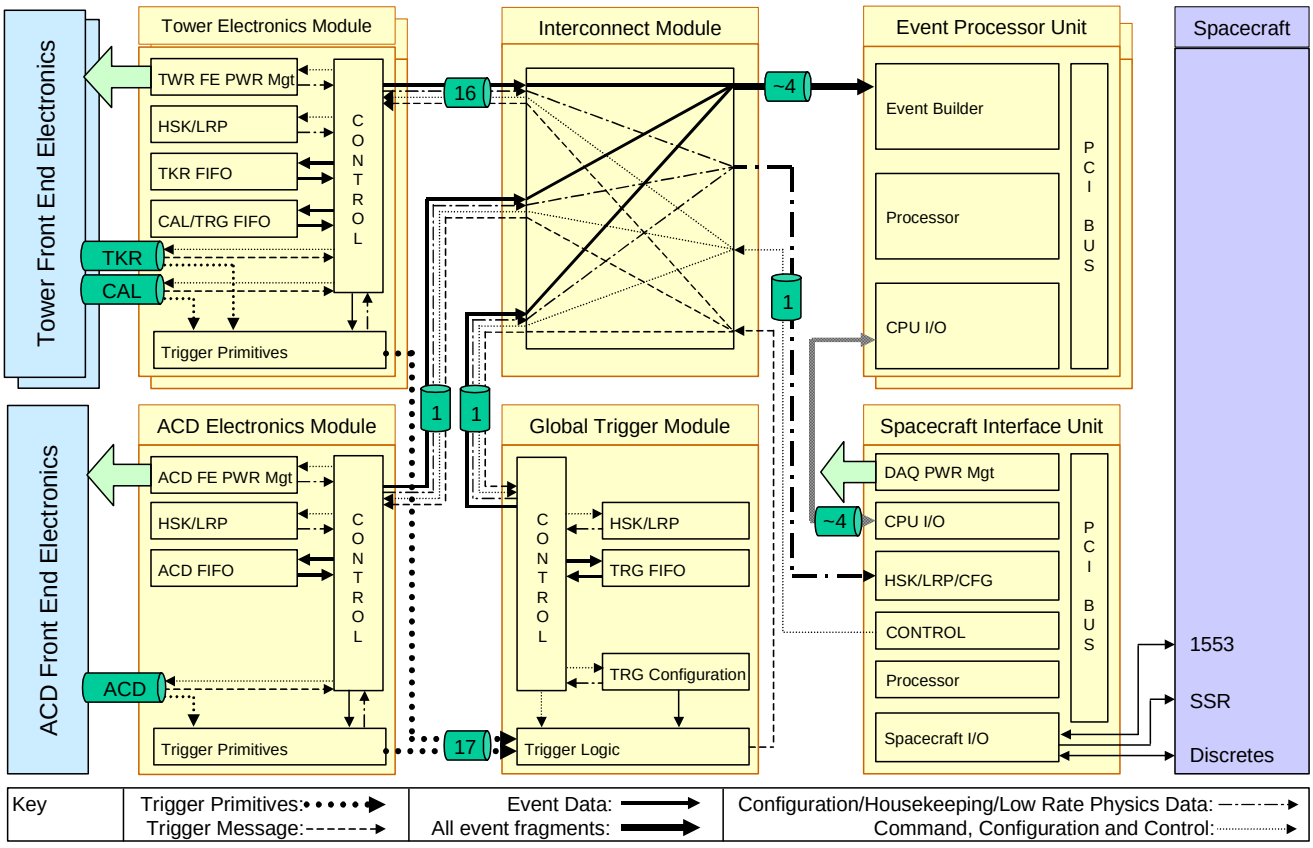

Figure 3 Trigger and Dataflow Block Diagram

3 Operating System

3.1 The Booting Process

The booting process anticipated for the LAT application is described. The model of a single control processor and four slave processors is used. The master processor is part of the SIU (Spacecraft Interface Unit). The slave processors are referred to as Event Processors (EP) and are part of the EPU (Event Processing Unit).

The booting sequence will proceed hierarchically, booting the master processor first. The master processor then serially boots each of its slave processors. The spacecraft is responsible for physically turning on and off the power to the SIU. The slave processor power will be switched either by the space vehicle (via a request sent by the SIU) or by the master processor.

3.2 The Master Processor

The following features, relevant to the booting process, distinguish the master processor from its slave processors:

- A larger Electrically Erasable Programmable Read Only Memory (EEPROM).
- Point-to-point serial links with each of its slave processors.
- A 1553 bus link to the spacecraft.

It is anticipated that the CPU boards to be used by the LAT will have insufficient EEPROM (~256 kByte) to hold all the LAT software. Because of this, a special board containing enough EEPROM (> 4 MByte) must be built. Due to physical constraints (space and power), only the master processor will have such a board. The master processor will serve any non-volatile data needed to the slave processors. For reliability, two completely functional EEPROM banks will be provided on each SIU.

3.2.1 Booting the Master Processor

Upon receiving a reset signal, a small bootstrap program in Read Only Memory (ROM) will begin execution. Placing this bootstrap code in ROM will ensure that it does not get overwritten. The ROM contains only the functionality that is absolutely necessary to initiate the boot process.

The bootstrap's first function will be to identify a functional memory block large enough to hold three memory segments:

- The bootstrap program data variables. The memory requirements here should be quite small.
- The executable memory image of the Real Time Operating System (RTOS) kernel.
- An upload buffer to hold a new RTOS executable sent from the ground across the 1553 bus.

A fourth possible memory segment would be a region to hold the main portion of the bootstrap code itself. Preliminary studies show that executing code directly from ROM incurs a large performance hit. A small ROM-resident portion of code would start the boot process. Once a functional boot region in Random Access Memory (RAM) has been established by an initial memory test, the remainder of the bootstrap program will be copied from ROM to the RAM code segment and executed from RAM.

The code for the RTOS kernel will be held in the EEPROM. Once a functional piece of memory is identified, the kernel will be transferred from the EEPROM to this block of memory. A Cyclic Redundancy Check (CRC) will be performed after transferring the code from ROM to RAM.

The kernel for the Real Time Operating System, VxWorks from Wind River Systems, comes in the form of a non-relocatable image. In order to provide protection against permanently bad memory occurring in the kernel's address space, a relocatable version of the kernel and a means of relocating it anywhere in memory will be provided.

This solution demands that a functional piece of memory the size of the kernel be located

somewhere in memory and that the bootstrap program be able to relocate the image. Writing a fully functional relocating linker/loader would be beyond the scope of this project. Preliminary studies indicate that a fully functional relocating loader is not necessary; a stripped down version should work.

There will be two bootstrap data regions designated ‘A’ and ‘B’ in the lower range of RAM memory addresses. The bootstrap program would start by testing region A. If all of the region A memory is functional, then this region is partitioned into the default boot memory segments. In this default case, all of boot region B will be overwritten when the RTOS kernel is started. If region A produces non-transient errors during the initial memory test, region B will be partitioned into the boot memory segments, and region A is ignored. This will most likely result in the loss of RAM equal to one boot region block for the processor when region B is selected.

The RTOS kernel image stored in EEPROM may also have more than one version. One image will be designated the default. This image is loaded when the bootstrap program proceeds autonomously with no command intervention. If no command or upload messages are received from the ground across the 1553 bus within approximately 1 minute of the bootstrap program start, then the default RTOS kernel image will be loaded and executed. Ground commands may specify an alternate kernel image.

To maintain the simplicity of the bootstrap program, the processor cache will be disabled, the processor external interrupts will be disabled, and the processor MMU will be disabled (physical addressing only). Any exceptions incurred during the execution of the bootstrap program will simply report the error and restart the bootstrap program from the beginning.

3.2.2 Master Processor Boot Commands

The master processor bootstrap program will contain the ability to process a small number of commands when manual intervention in the boot process is required:

- A no-operation command to check the command and telemetry links.
- Restart the bootstrap program.
- Load and execute a specific RTOS kernel image.
- Commit a newly loaded RTOS kernel image from the RAM upload buffer to a location in EEPROM.
- If necessary, power control commands.

3.2.3 Master Processor Boot Uploads

The master processor bootstrap program has the ability to receive an uplinked RTOS kernel image from the ground across the 1553 bus. The upload is buffered in a special RAM boot memory segment until the file is complete and its integrity verified. Once the upload is complete, a ground command is necessary to commit the new kernel image to a location in EEPROM. The logistics of the kernel relocation and load process may make loading a new kernel image directly from the RAM upload buffer infeasible.

3.2.4 Master Processor Boot Telemetry

The master processor bootstrap program will report a limited set housekeeping telemetry. This will include a timestamp, command and upload execution status, error reports, and any critical environmental monitors.

3.2.5 First Level Application Booting

Once the RTOS kernel is booted, the software driver for the 1553 bus interface is loaded. This establishes an intelligent communication link with the spacecraft as early as possible, opening a channel to report the progress of the remainder of the boot sequence. At this point, the master processor begins reporting its full set of housekeeping telemetry. The full command and upload receive interface is then started. Once complete, the master processor periodically assembles the full set of instrument real-time telemetry for downlink on the 1553 bus. Otherwise, the master processor listens on the 1553 bus for new command or upload messages and processes those messages as needed. The master processor will also load the boot and file server for the slave processors as well as the driver for the CPU point-to-point links. At this point, the slave processors may be powered on and booted.

3.3 The Slave Processors

Each event processor has a small piece of ROM containing its bootstrap code and code to support a point-to-point serial link with the master processor. The booting sequence of the event processors is similar to the master processor. Much of the code from the master processor bootstrap program should be in common with the slave processors.

3.3.1 Booting the Slave Processor

The bootstrap code must first identify a functional piece of memory to load the RTOS kernel. Like the master processor, the slave processors require three good memory segments: bootstrap program variables, a kernel execution area, and a kernel image file upload buffer. Instead of loading the code from EEPROM or across the 1553 bus as the master processor does, the slave processor bootstrap code must initialize and establish communications over its serial link to the master processor. Once communication with the master processor is established, the RTOS kernel can be loaded from the master processor over the serial link, into the slave's memory. As in the master processor, a cyclic redundancy check is made when transferring the software into RAM.

The slave processor bootstrap code contains the same relocating loader as does the master processor. The bootstrap program may load the RTOS kernel into an alternate memory region. After loading and relocating the kernel code, control is transferred to the RTOS's startup code.

3.3.2 Boot Times

The boot time of the slave processors will be limited by the serial links. These links are anticipated to be capable of 20-40 Mbits/second. Moving software images in the 1-2 MByte range over the links will take 0.5 to1.0 seconds. It may be possible to cache the RTOS kernel image file in the slave processor local RAM. This would allow the slave processors

to warm reboot without needing to reload the kernel from the master processor.

3.3.3 Slave Processor Boot Commands

The slave processor bootstrap program will be able to process a small number of commands when manual intervention in the boot process is required. These commands are forwarded to the slave processor by the master processor.

- A no-operation command to check the command and telemetry links.
- Restart the bootstrap program.

3.3.4 Slave Processor Boot Telemetry

The slave processor bootstrap program will report a limited set housekeeping telemetry. This will include a timestamp, command and upload execution status, and error reports. The slave processor telemetry is sent on the serial link to the master processor.

3.3.5 First Level Application Booting

Once the RTOS kernel is booted, a more sophisticated driver for the point-to-point serial link to the master processor is loaded. At this point, the slave processor begins reporting its full set of housekeeping telemetry. The full command and upload receive interface is then started so that alternate configurations and parameter tables may be loaded. The remaining instrument application software modules are loaded from the master processor's EEPROM and full instrument operation may commence.

3.4 Warm/Cold Boots

The system will provide the concepts of both warm and cold boots. Warm boots restore the system as closely as is possible to the state it was when the reboot was initiated. Cold boots always restore to a default configuration. In order to support a warm boot, the current operating configuration must be maintained in non-volatile memory.

3.5 RTOS Kernel Services

3.5.1 Introduction

The RTOS kernel will provide a number of very low-level services. For instance, it will provide basic interrupt and input/output management processing. It will use and manage the on-board local RAM for the storage of executable programs, program constants, temporary data and I/O processing functions. It will use on-board non-volatile memory for storing persistent software modules and configuration information.

3.5.2 Task Scheduler

The RTOS kernel controls the scheduling and execution of software tasks. The scheduling algorithm is preemptive and rigid: higher priority tasks are always scheduled ahead of lower priority tasks if the higher priority tasks are ready to run. The RTOS kernel provides synchronization primitives (semaphores and locks) both to protect shared resources and to

provide signaling between two tasks. The RTOS also manages the low-level details of handling hardware interrupts. Interrupts may signal tasks to indicate that a new event of interest has occurred.

3.5.3 Memory Management

The RTOS kernel maintains a pool of available volatile memory both for its own use and for the use of applications and device drivers. At startup, all of the memory available beyond the kernel image itself is assigned to the system pool. The management of the system memory pool will include the following functionality:

- The ability to allocate a block of arbitrary size from the system pool.
- The ability to return a previously allocated block to the system pool.
- The ability to report to the ground the amount of free memory remaining in the system pool and the level of fragmentation that has occurred in the system pool.

Applications and drivers are free to allocate large blocks of memory from the system pool, and then implement more efficient private memory managers within that block.

3.5.4 File Management

The kernel will provide the capability to manage files in EEPROM. Various types of persistent data storage are required such as software object modules, instrument configuration and calibration tables, and command lists and histories. This management responsibility includes:

- The ability to load a new file from the ground.
- The ability to remove files when space becomes an issue.
- The ability to identify the current collection of files stored in a directory. The report will include the modification times and the sizes of the files.
- The ability to identify the current set of directories. To make this task more manageable, the depth of directories will probably be limited to one or two.
- The ability to dump the contents of a file into the data recorder.

All files will have a timestamp indicating the last creation or modification date.

From a functional viewpoint, these services treat the EEPROM as a disk, managing the software modules much as a standard OS manages files on a disk.

3.5.5 Exception Processing

The kernel will contain exception-handling code. The primary purpose of this code is to provide a reporting mechanism and an orderly shutdown of the system when an exception occurs. The information from the reporting mechanism will be used to diagnose the cause of the exception.

In a limited number of cases, the shutdown and restarting of individual tasks may be possible. However, given that the VxWorks kernel provides no memory protection across

task boundaries, the integrity of the processor is in doubt once an exception has been fielded. Also, given the interconnectedness of many of the tasks, it may prove impractical to shutdown and restart a single task. The easiest and safest strategy is likely to initiate an orderly shutdown and request that the processor reboot itself. The strategy is to accept a complete reboot as a fact of life, but to make that process as quick and painless as possible.

3.5.6 Watchdog Facility

In addition to handling hardware and software exceptions, a watchdog facility will be provided to ensure that the processor is performing useful work. This mechanism must work without ground intervention. A multi-tiered strategy is used.

Keep-alive messages between the master processor and its slaves provide the first line of defense. When the master fails to receive a keep-alive message from a slave it tries the following two actions:

- Send a reboot request message over the point-to-point link to the slave.
- If that does not work, reboot the processor by tugging on its reset line.

The first is meant to be a 'gentle' request, allowing diagnostic information to be gathered before the actual reboot is initiated. The second method is to be used only when the slave processor fails to respond to a reboot request.

Neither of these solutions provides protection when the master processor itself is hung. To protect the master processor and to act as a final line of defense, each processor board should be equipped with a hardware watchdog timer. If the software fails to reset the timer before it expires, the watchdog timer pulls on the processor's reset line. The strategy is to reset the timer if and only if useful work is being performed.

3.6 IO Services

Once the kernel is booted, software modules supporting the various IO ports are loaded and initialized. The services provided by this software will be in the form of shareable libraries, interrupt service routines and/or tasks. This section only describes the functionality that these services provide, not their specific usage.

3.6.1 1553 Interface

The LAT software provides services that control the 1553 bus interface. Currently only the master processor has a 1553 bus interface. The polling frequency is both statically set at boot time and dynamically configured in response to changing conditions.

3.6.2 Solid State Recorder Interface

The LAT software provides services that control the writing of data to the Solid State Recorder (SSR). In addition to this basic write functionality, any ability to control the SSR and any global information about the SSR, such as space left, will be exposed through this software.

Only the master processor has an interface to the SSR.

3.6.3 GBM Immediate Alert

The LAT software provides services which field a GBM Immediate Alert message. This message is in the form of a single discrete signal that will be raised within 5msec of the GBM recognition of a GRB candidate. The processor sees this signal as an interrupt source.

Only the master processor has an interface to the GBM Immediate Alert Signal.

3.6.4 Electronics Commanding And Readback Interface

The LAT software provides services that control the Electronics Command and Readback Interface. The command interface provides the path to initially configure the electronics modules (TEMs, AEMs, GLTs). For example, each strip of the TKR has 2 mask bits associated with it to remove noisy strips from the trigger and/or data. These bits are configured via the electronics command interface.

All quantities configured by the command interface can be non-destructively read back via the Read Back Interface allowing verification of the configuration. Housekeeping and Low Rate Physics data from the electronics modules also arrives on this path.

A close coupling exists between the Command Interface and the Readback Interface. Requests to read back configuration data are initiated on the Command Interface, with the data arriving on the Readback Interface. The LAT software must provide the necessary coupling between these two IO ports to form an effective round trip path. The hardware allows only one command at a time to be issued by enforcing a minimum time between commands. The software has complete responsibility for providing flow control on the readback path. A readback command will not be issued until the data from a previous read back command has been received.

This functionality exists only on the master processor.

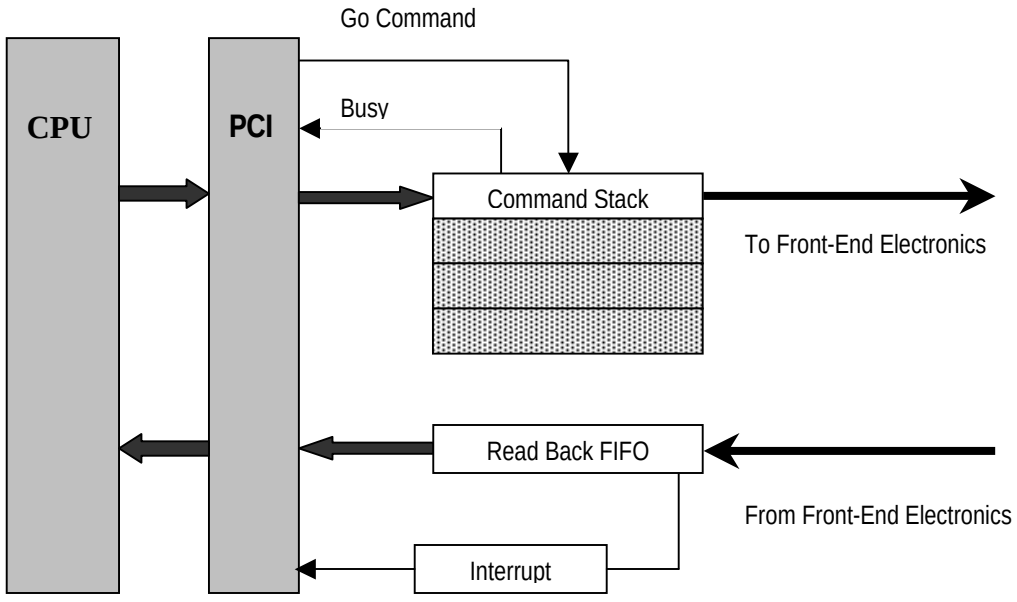

Figure 4 Block Diagram of the Command and Readback Path

3.6.5 Inter Processor Communications

The LAT software provides services that control the inter-processor communication links. These links exist as point-to-point links between the master processor and each of the slave processors, but not between slave processors. Services are provided to support direct master to slave communication. The master processor supports third party transfers on behalf of the slaves, enabling a form of slave-to-slave communication.

Although it is not anticipated that both the master processor and its redundant counterpart are running simultaneously, a communication link is also provided between the master processor and its redundant counterpart. This will facilitate the transfer of software between them, avoiding uploading of the software twice.

Both messaging services and bulk data protocols are to be supported.

3.6.6 GPS Services

The LAT software provides services that support the handling a GPS timing interrupt and the reading of a GPS message. This service will be tightly connected to various LAT timing and clock services.

3.6.7 Electronics Modules Input Data Services

The LAT software provides services that control access to the event data originating from the front-end electronics. The DAQ hardware DMA's events across the PCI bus into a dedicated section of the processor's memory configured as a ring buffer.

The input data services must

- Handle the DMA interrupts announcing the arrival of data.
- Control software access to the data present in the ring buffers.
- Provide the ability to notify the DMA hardware when data has been consumed so that memory may be freed.

Under one proposed design, each slave processor must configure an acceptance lookup table that controls which events that slave processor is willing to handle. The contents of these tables is coordinated by the master processor in order to ensure that all events are handled by at least one slave processor. In another design, there is one central table that controls the event dispatching. In this design, the master processor would be responsible for loading the central table.

4 Event Processing

This section describes the receipt, filtering, processing and packaging of physics events from the instrument. From a scientific viewpoint, this is one of the most critical functions the flight software performs. From a hardware viewpoint, the event processing is the driving factor when determining the number of CPU cycles.

4.1 Nature of the Data

The instrument produces a packet of data each time it is triggered. This data consists of 18 blocks, 16 representing the data from each of the tower electronics modules (TEMs), 1 from the ACD electronics modules (AEM) and 1 from the Global Trigger (GLT). The dataflow hardware (called the Event Builder) assembles these pieces into one data structure, called an *event*. The event is then directed to at least one EP for processing.

4.2 Reception Of Events

The LAT software services controlling the electronic modules input data notify the application data processing software of the arrival of data. This notification will be by a message describing all the events currently available for processing.

4.3 Integrity Checking

The LAT software checks the integrity of the events as they arrive. This checking is limited to verifying that event has been properly constructed and transported to the processor. In particular, data quality is not checked at this point. The software is configurable to provide different levels of checking on selectable sub-samples of the data. For instance, this could allow exhaustive checks on a small portion of the events and rudimentary checks on all events. Resource usage and need sets the proportions

4.3.1 Completeness

The event will be checked to verify that all 18 components are present and that data from the 16 tower components has both a TRG/CAL section and TKR section present.

4.3.2 Correctness

All components are checked to verify they are from the same event. Two quantities are used in this test. The first is an identifying event number assigned to each component as near to its point of origin as is possible. In practice this is a sequential number, advanced with each trigger. In addition, the value a local clock is captured. At this point in the design it is not known whether all clocks will be synchronized by the hardware. If the clocks are not synchronized by the hardware, the LAT software will provide a means of synchronization.

To pass this test, all 18 components must have exactly the same identifying event number and must be consistent with having been acquired at the same time.

4.3.3 Front End Integrity

The above checks are directed primarily at spotting errors that are introduced during the transport of event fragments from the electronics modules to the event builder and the subsequent transfer of coherent events into CPU memory. In addition, the front-end electronics also attaches error information to detect problems that occur in transporting and assembling the data from the front-end electronics to the electronics modules. The nature and checking of this information is now described.

4.3.3.1 CAL Front End Integrity Checking

The Calorimeter data has a number of parity bits to detect bit errors that occur during the transport over the internal serial links. Any Calorimeter errors occurring on a tower result in the setting of a summary error bit. Events with the summary bit set are examined in detail.

4.3.3.2 TKR Front End Integrity Checking

The Tracker has, in addition to the parity bits, a collection of numbers carrying the low 2 bits of the event identifier. Each TKR GTFE chip attaches these bits to the data when the hit strip data is first latched into the one of its four front-end buffers. When the data is read out, the hardware compares this tag with the expected number. Any discrepancy is summarized, allowing suspect events to be further examined by the software. This protects against various problems, for example when a GTFE does not correctly advance to the next front-end buffer. Note that the hardware is constructed such that error information is reported *independently* of whether the GTFE has any data.

4.4 Event Filtering

The Event Filtering stage is a software filter, designed to reduce the hardware triggered event rate (up to 10 kHz) to the 30-60 Hz rate of gamma candidate events. The very high input rate demands that the filtering software be very efficient. The high rejection rate demands a reasonably sophisticated process be used to monitor the efficiency and purity of the filtering algorithm.

The event filtering process is the determining factor when sizing the processor power. However, it has very little impact on the memory requirements. The relatively small size of the events, anticipated to be < 2 kBytes, means that the processor's level 1 data cache will be very effective. Preliminary studies also indicate that the filtering code itself is sufficiently compact to be contained within the processor's level 1 instruction cache.

4.4.1 Event Classification

Before an event is subjected to the filtering process it is first classified as either a *candidate* gamma ray event or a heavy nuclei (CNO) event. The CNO event rate is expected to be around 50 Hz. These events are identified by the presence of an ACD tile with a HI discriminator bit set. Such events are placed to the side for further processing. A large fraction of these will be sent to the ground to aid in calibration. The remaining events are gamma ray candidates and are passed on to the next stage of filtering.

4.4.2 Filtering of Gamma Ray Candidates

The presence of a large amount of energy in the Calorimeter indicates events that most likely will be kept. A bit in the global trigger identifies these events so that they can be quickly identified. The fraction of such events is small, anticipated to be 10-50 Hz out of a total rate of >5 kHz. Algorithms suggested by the CAL group efficiently reduce this sample to a 1 Hz rate.

4.4.2.1 Charged Track Rejection

The bulk of the events are protons that traverse the LAT. The LAT filtering software attempts to identify these by using the silicon strip data to determine the presence of a track and then to associate that track with a coincident hit in the projected ACD tile. Studies show that this technique reduces the event rate by about a factor of 6. Studies on Monte Carlo Events and beam test data show that this association can be made very quickly (~20 usecs on a PPC 603 processor running at 150 MHz).

The remaining events are good gamma candidates. The sample is still too large to accept, but the rate is now sufficiently low that more sophisticated algorithms can be used. Offline studies indicate that these algorithms are very effective at reducing the sample, preserving good gamma efficiency while achieving good purity.

The necessary purity of the sample is driven primarily by the output bandwidth. The more output bandwidth, the less discriminating the onboard filtering needs to be. This is highly desirable, because it is inevitable that more discriminating onboard filters will cost efficiency and robustness. Once the onboard filter rejects an event, it is permanent.

4.4.3 Monitoring

Because the finality of the filter processing decision is absolute, a means of monitoring the efficiency and purity of the events that pass the filtering criteria is necessary.

Ground software can easily monitor the purity of the sample by using sophisticated algorithms not available to the flight software operating in real time on the data stream. Provided the resulting data volume can be shipped to the ground, the consequences of an impure sample are not severe. The main exposure comes from trying to use these events when doing in-flight science, such as GRB identification. All events will be assigned a quality factor. The quality factor for events used in on board science applications will be set higher than for those shipped to the ground, thus sacrificing efficiency for purity.

Monitoring the efficiency is a difficult process due to the very low acceptance rate, potentially only 30/10000. The usual tactic of leaking a fraction of unbiased events into the output stream is thwarted by this acceptance ratio. To illustrate, a sample of 300,000 unbiased events yields 100 gamma ray events, enough to do a 10% measurement. This data sample is the equivalent of 3 hours normal running. Often overlooked is the need to monitor this efficiency as a function of some other parameter. For instance, it is inevitable that the filtering efficiency will be a strong function of the gamma energy, particularly at the lower energies (< 500 MeV). Subdividing of the data to discover these dependencies only serves to further aggravate this problem.

4.5 Further Processing

Candidate events may be prioritized for shipment to the ground. This allows more effective use of the bandwidth. However, finite buffering within the processor and limited functionality to the SSR may make this impossible to do.

No further *on board* processing is needed for use of these events by the ground science team. To be useful for doing in flight science, such as GRB or AGN detection, more processing may be needed. For instance, the photon direction and energy may need to be

refined and event timing information examined to establish the time evolution of a transient.

4.6 Compression

Candidate events will be reformatted to make the best use of the available downlink bandwidth. This will include eliminating obviously redundant information. For example, all event identifiers can be replaced with one event identifier and a summary bit when all event identifiers match. Various packing and compression techniques will also be used to reduce the size of the event. The compression techniques may be loss-less (for example Huffman encoding) or may be lossy (for example, more aggressive threshold suppression in certain cases). Preliminary studies give a compression factor of 2, effectively doubling the output bandwidth available.

4.7 Packaging

Output events will be packaged in CCSDS format. The spacecraft vendor will then provide Reed-Solomon encoding of these packets and the necessary randomizer to prevent frequency spikes in the downlinked signal. The Reed-Solomon encoding may occur before or after the storing the event in the SSR. Applying the encoding before storing the event will inflate the event by approximately 25%. This must be considered when sizing the SSR.

4.8 Miscellaneous

The event-processing path will allow the transport of the full data stream directly to the output. Naturally, due to limited space on the SSR and limited downlink bandwidth, this will normally only be done for diagnostic reasons. The 32 Mbit/sec link to the SSR was chosen specifically to support such direct transfers during diagnostics and to provide capacity for transient events, such as during a GRB or AGN.

5 Commanding

The command interface with the vehicle is over the MIL-STD 1553 bus. The exact command format and protocol is TBD, but it is expected that there will be a short authenticated command type and a longer “block” command used for uploads. The authenticated command type is used for real time commands.

5.1 1553 And Commanding

All commands arrive from the spacecraft from the 1553 bus in a message packet. The flight software will first extract the command from any jacketing protocol before any further processing.

Two 1553 subaddresses are used for commanding: one for real time commands and one for block commands. Any command arriving at the real time command subaddress is executed immediately, regardless of the state of command reception on the block command subaddress.

5.1.1 Real Time Command Reception And Execution

Commands arriving at the real time subaddress of the SIU are first checked for correct parity or CRC (TBR) and then for general command validity. If the command passes all tests the command accept counter is incremented and the command processed. If the command fails any of the tests the command reject counter is incremented and the appropriate error code issued.

If the destination of the command is the SIU it is then executed. If the command has a destination other than the SIU the command is forwarded the appropriate destination and the command forwarded counter is incremented.

5.1.2 Block Command Reception And Processing

All commands that are not real time are considered block commands and are sent to the block command subaddress. Block commands to the LAT are subdivided into transfer blocks for transmission. The size of the transfer block will be dependent on the actual implementation of the system but will most likely be ~1-2 kBytes. This is a balance between the time for setup and the time for transmission. Bigger blocks are more efficient to transmit, but have higher overhead for retransmission/error recovery. Each transfer block will contain the command number and the sequence number giving the position of the transfer block within the command.

Upon reception of the first transfer block in a command the system allocates a command reception buffer and after checking the transfer block CRC places the block in the correct location in the buffer as determined by its sequence number. It is not anticipated that blocks will be transmitted out of sequence, but it is possible that there will be a transmission error on a block that is not noticed until the transmission of the next block has begun. This technique will allow the full sequence of transfer blocks to be sent as fast as possible and the selected blocks to be retransmitted as required to fill in blocks lost to errors. Upon receiving the last transfer block of a command the system will verify that all blocks have been received and issue an error report identifying any missing blocks. If the sequence is complete the system then checks the command CRC and, if that passes, continues to process the block command as is appropriate for the type of block.

The system will be designed to allow two block commands to be in the process of transmission simultaneously. Practical experience says that ground contacts do not always go as planned and that some uploads require more than one contact to complete. To cope with these situations, it is desirable that the system allow a large upload to be interrupted with a smaller upload, such as daily time tagged commands.

5.1.2.1 Block Command Types

The block command types fall into three broad categories:

- Executable code uploads.
- Table/parameter uploads.
- Command uploads.

5.2 Command Storage And Execution

The LAT flight software command system shall store up to TBD commands on board for later execution. Stored commands shall have a maximum length of TBD bits. Storage time shall be at least 48 hours with an execution resolution of 1 second. Real time commands shall have a higher execution priority than stored commands. Stored commands shall be executed within 10 milliseconds of its schedule time by the LAT flight software system.

6 Housekeeping

Housekeeping is the process of monitoring the health of the LAT. It is performed independently of the physics event acquisition and the results stored separately. Housekeeping is a continuous activity requiring only that the SIU be powered and booted.

6.1 Acquisition Of Housekeeping Data

Housekeeping data is solicited and consumed by the SIU. The SIU controls the LAT configuration and thus knows what segments are capable of interrogation. Once the LAT is fully powered, the SIU will collect housekeeping packets from:

- Power Distribution.
- Each TEM (sixteen packets).
- The operational AEM.
- The operational GLT.
- Each booted EPU.
- Itself.

The SIU will interrogate the LAT at 1 Hz using the command path, and is responsible for validating the integrity of each of the returned packets received across the CFG/HSK/LRP path.

The data content of a housekeeping packet varies by target. DAQ electronics boards (TEMs, AEMs, GLT, power distribution) typically report voltages, currents and temperatures. A housekeeping packet from a processor board (SIU or EPU) will contain more extensive information including memory usage, processor idle time, status of I/O channels and other resource information.

6.2 Analysis Of Housekeeping Data

The SIU is responsible for the first pass analysis of the housekeeping data. For simple variables (e.g. voltages) the SIU keeps histograms and trend graphs. Such variables can be provided with warning and error limits so that a variable returning a warning value for more than (TBD) housekeeping cycles, or returning an error value will cause the SIU to initiate real time ground notification. For more complicated variables, the analysis principles are similar though the algorithmic treatment of the variable(s) may be more involved.

For a limited number of well identified cases, housekeeping analysis may initiate

autonomous corrective action. Example: reducing ACD high voltages should unprotected entry to the SAA be indicated by high currents in the ACD (see section 10 “Transiting the SAA”). Any such actions will be reported in the normal telemetry stream.

6.3 Disposal Of Housekeeping Data

The volume of housekeeping data as currently defined is sufficiently small (less than 10% of the available downlink bandwidth) that all the housekeeping data can be stored to the SSR for later transmission to the ground.

7 Low Rate Physics Data

Low rate physics data consist of scalers maintained in the front end electronics. In many respects these scalers look like housekeeping data, but have the advantage that the counting interval is carefully timed and is synchronized across all scalers. The SIU is responsible for soliciting and processing LRP packets. The LRP activity is independent of physics event acquisition and interacts with housekeeping only to the extent that they share the same command and data acquisition paths. The number of scalers is limited as follows:

Subsystem	Element	Scalers/Element	# Elements	Total
ACD	AEM	12	1	12
CAL	TEM	2	16	32
TKR	TEM	2	16	32
Trigger	GLT	4	1	4
Grand Total				80

Table 6 Estimate of the number of Low Rate Physics Scalers

While the number of scalers is limited, each scaler can be multiplexed by command to many sources allowing a wide variety of correlated measurements to be made.

7.1 Acquisition Of Low Rate Physics Data

The SIU interrogates the LAT for LRP packets at a rate of (TBD, but in the range 0.1 to 100) Hz using the command path, and is responsible for validating the integrity of each of the returned LRP packets received across the CFG/HSK/LRP path.

Acquisition of LRP data differs from housekeeping data in that the SIU first issues a “latch” command which captures the current state of the scalers (all scalers in the LRP system are double buffered) and then issues read commands to capture those values.

7.2 Analysis Of Low Rate Physics Data

On-board analysis of LRP data occurs on the SIU. The primary use of LRP data will resemble housekeeping analysis: the front end scalers will be multiplexed around a list of sources and the rate of each source compared to warning and error limits. A secondary use of the LRP data, invoked by command, is to pursue diagnostic tests looking at specific sources.

For a limited number of well identified cases, LRP analysis may initiate autonomous corrective action. Example: reducing ACD high voltages should unprotected entry to the SAA be indicated by high hit rates in the ACD (see section 10 “Transiting the SAA”). Any such actions will be reported in the normal telemetry stream.

7.3 Disposal Of Low Rate Physics Data

Low rate physics data is not voluminous and can be recorded in its entirety for later transmission to the ground.

8 Instrument Configuration And Control

The state and behavior of the LAT is controlled by a large number of switches and registers in the front-end electronics. The complete assembly of such state information constitutes a configuration. The LAT is designed to be configurable for a variety of reasons. The following are examples:

- Scientific Objectives. Different scientific objectives require different modes of operation. Each mode is supported by a configuration.
- LAT Health. Transiting the SAA requires that the instrument, specifically the ACD, be protected from radiation damage. This requires an “SAA configuration” with reduced ACD high voltage.
- LAT Functionality. In the event of component failure the configuration can be trimmed to exclude the broken component, with only a small loss of LAT capability. Example: a noisy tracker strip can be excluded from the trigger.
- LAT Redundancy. In a variation of the above, if the component can be bypassed on a redundant path then the broken component can be compensated for with no loss of LAT capability. Example: switch from global trigger box A to global trigger box B.

The point of control for configurations is the flight software. Flight software must be able to maintain and manage a number of configurations. It must be able to read, write and transfer configuration information across three interfaces:

- The on-board EEPROM (using the file system outlined in section 3.5.4 “File Management”). This provides the non volatile configuration storage.
- The front-end electronics (using the commanding system provided by the DAQ).
- The ground (to upload new configurations and to save configurations needed for ground analysis).

8.1 Configuration Arithmetic

A complete, static description of the LAT comprises a large volume of data:

Subsystem	Element	Bytes/ Element	# Elements	Total (bytes)
ACD	GAFE	4	6 (boards) * 2 (redundancy) * 18 (channels)	864
	GARC	14	6 (boards) * 2 (redundancy	168
CAL	GCFE	5	16 (towers) * 192 (logs ends)	15360
	GCRC	8	16 (towers) * 16 (GTRC/tower)	2048
TKR	GTFE	26	16 (towers) * 36 (layers) * 24 (GTFE/layer)	359424
	GTRC	2	16 (towers) * 36 (layers)	1152
TEM	Registers	2	16 (towers) * 44 (registers)	1408
AEM	Registers	2	1 * 20 (registers)	40
GLT	Registers	2	8 (triggers) * 6 (registers)	96
Grand Total				380560

Table 7 Estimated size of a full detector configuration description.

For this reason a more compact representation will be needed to store configurations. There are two major possibilities:

- Variation from a standard. This involves defining a “standard” configuration and then recording only those elements that do not correspond to the standard. Example: assuming that the standard is that all tracker strips contribute to the trigger, then keep a file listing the tracker strips which must be removed from the trigger.
- List of transactions. In this method a configuration is characterized as the position reached as a result of a list of commands. (Note that the list must make no assumptions about the beginning state of the configuration and should start with something like a “reset” command that establishes a known, fixed … though potentially not very useful … configuration). Repeating the example in the previous bullet: keep a list of commands where the first command enables all strips into the trigger, and each subsequent command removes a tracker strip from the trigger.

In the example given, the former is probably more compact and is certainly a more convenient format for ground software to deal with. On the other hand, the latter corresponds more directly with the way that flight software must establish a configuration, i.e. by issuing commands to the front end electronics. Some compactness can be restored to the latter by allowing the transactional description to use meta commands which flight software then expands into the available physical commands. For example, a meta command might read (in the binary format chosen) “reset all”, to which the flight software responds by issuing sixteen tower, two ACD and one global trigger reset commands.

8.2 Operational Use Of Configurations.

The operational use of configurations can be conveniently divided into physics and non-physics running.

8.2.1 Non-Physics Running

Throughout the lifetime of the LAT a variety of diagnostic and calibration tasks will need

to be run. It is hard to predict what configurations will be needed for the former (especially in cases where the diagnostic is being run in response to a fault condition) and the latter frequently cycle through a series of configurations. Thus neither case is a good candidate for using static configurations stored in EEPROM. To handle these cases, flight software must provide low level utilities to allow programmatic manipulation of the configuration along with the ability to record (for later transmission to the ground) the configuration currently loaded in the front end electronics.

8.2.2 Physics Running

In physics running, the goal of configuration management is to establish and verify a configuration that matches the physics goals, and to then keep that configuration fixed for as long as the observation is in progress. This is a good candidate for prepared configurations stored in EEPROM. Such configurations can be developed and loaded before launch, but changes to these configurations should be expected in response to either unexpected physics conditions or LAT instrument "decay". The configuration extant during an observation is a vital element of the data analysis and flight software should record and transmit the configuration in effect for each observation run.

8.2.2.1 Spontaneous Loss Of Configuration.

While keeping the configuration constant for the duration of an observation is the goal, it cannot be guaranteed. The following are examples of causes of loss of configuration

- A CPU could take an exception due an SEU.
- A tower power supply might fail.

Where command and control are still available (i.e. the CPU taking the exception is not the SIU), the flight software response should be to bring the current observation to as orderly a halt as possible. Strategy thereafter will be a function of the fault. Fault modes are at this point difficult to enumerate, so the basic philosophy is "to do no harm". Flight software will move to a "safe" mode and await ground command. For identified and well understood cases, it may be possible to develop restart procedures. Example: if an event processing CPU takes an exception, it may be within the SIU's competence to reboot it and restart the observation.

8.3 Autonomous Update Of Configurations.

The bandwidth of the telemetry link between the spacecraft and the ground is severely limited. Thus collecting non-physics calibration/diagnostic data for ground analysis comes at the cost of reduced physics data. Such scenarios are probably inevitable, but the resources available to flight software are such that some non-physics and diagnostic operations can be carried out in situ. Example: program a commandable procedure to enable all tracker strips into data acquisition, trigger the detector ~100k times, then analyze strip populations to produce lists of dead and noisy strips. The resulting lists could be stored in EEPROM and used as part of the physics observation configurations.

This constitutes an autonomous update of the configurations stored on board. Such a procedure provides several advantages:

- Little impact on observation time.
- Little impact on telemetry bandwidth (such autonomously produced lists should of course be downloaded).
- Fast turnaround between the construction and use of the information (if it's determined that there is no need to "put a human in the loop").

9 Monitoring and Calibration

9.1 Introduction

This section describes monitoring and calibration procedures that take place both inside and outside the context of normal data taking. Whenever possible, these procedures are interleaved with normal data taking.

The emphasis is on procedures that cannot be done using the data sent to the ground. For instance, the abundant source of cosmics that pass through the detector provides a known input that should result in a predictable output. These are ideal candidates to monitor detector performance and which are not available in such large quantities to the ground.

Some procedures discussed in this section are incompatible with data taking because they change the configuration of the instrument. They have as goals:

- Verifying the integrity of the data path, including the Global Trigger.
- Calibrating the performance of the ACD, CAL and TKR subsystems by injecting a known charge on the signal path.

Because calibrations procedures elicit a known response from the instrument, these two goals become intermingled. Calibrations often provide the first sign of a developing problem and serve as the last step in proving that a problem has been addressed.

9.2 Resource Usage

9.2.1 Monitoring

The monitoring and calibration procedures place heavy demands on DAQ and CPU resources. Monitoring the large number of channels and wide dynamic ranges present in the pulse height information requires a large amount of storage. In particular, monitoring activities that occur during normal data taking are likely to be the main drivers when determining overall memory sizes. A rough figure of merit indicates that each of the three major subsystems (ACD, CAL and TKR) will need approximately 10Mbytes of memory for such monitoring. CPU boards under consideration typically have 64-128MBytes.

CPU usage for these activities must also be considered, although here one can trade CPU time against statistics by prescaling or filtering the events used in monitoring activities.

Controlling memory needs is harder; there is no simple prescaling or priority scheme. There are two ways to reduce the memory needs.

- Place different monitoring tasks on different CPUs.

- Only monitor a fraction of the detector at any given time.

This first strategy has two downsides. The first is statistics; if events are normally directed at only one CPU, then that event will only contribute to the monitoring occurring on that CPU. The second is a reliability problem. If a CPU is lost as the system ages, the monitoring activity that CPU was providing must be moved to another CPU. One can compensate for the increased CPU load by reducing the number of events being monitored, but the memory to store the results cannot be replaced.

This leaves one with the second strategy. One will be forced to monitor sections of the LAT detectors, rotating amongst the target sections when enough statistics have been accumulated.

9.2.2 Calibration

Calibration procedures can impose a very different set of requirements than normal data taking and need to be considered when sizing DAQ resources. Incorrect sizing of the DAQ pipes from the front-ends to the CPUs could result in very long calibration times. Worse, calibration event data may exceed size limitations in the DAQ pipes that were set by considering *sensibly* sized events making it difficult or even impossible to transport calibration events. The LAT DAQ architecture is such that, while such limits are imposed, it is always possible, with some performance penalty, to calibrate the instrument.

9.3 Background Contamination During Calibration

Operationally, one must decide whether to calibrate with the sensors on or off. Turning the bias voltages off in the CAL and TKR leads to large capacitances and, therefore, higher noise. For the ACD, it means leaving the high voltage on. Leaving the sensors active will contaminate the calibration triggers with background events. The level of contamination is determined by the background rate, shaping/memory times and granularity of the detectors.

Preliminary estimates indicate that this is potentially only a problem with the ACD.

9.4 ACD

9.4.1 ACD Introduction

The ACD data consists of two sets of 89 discriminators (one primary, one for redundancy) and pulse height information that is zero suppressed and auto-ranged. The zero suppression can be disabled on selective triggers for monitoring purposes. The ACD electronics provides no way to force both ranges to be simultaneously present in the data stream.

The discriminators must be set at a threshold commensurate with being highly efficient for a minimum ionizing particle (MIP) while staying away from the noise. Two other sets of discriminators are present in the ACD, but are not directly recorded in the data stream. One set of discriminators is set at a higher threshold than 1 MIP and seeks to identify CNO (Carbon, Nitrogen, Oxygen and higher Z) events. The other set determines the zero suppress threshold used by the pulse height data.

A limited set of rate counters is available to monitor counting rates in each of the ACD

tiles. Each ACD tile can be selected to be monitored by connecting it to a rate counter via a multiplexer.

The goal of ACD monitoring and calibration is to

- Determine the proper DAC thresholds for each set of discriminators.
- Monitor the efficiency and noise characteristics of each ACD tile.

Charged tracks, both protons and CNO events, are used to set an absolute scale. High precision DAC's can inject a known amount of charge directly into the front-end pre-amps, effectively calibrating the electronics gain of the system.

9.4.2 ACD Monitoring

During running, the DAQ will maintain the following information

- Simple population maps for each discriminator.
- Pulse height distributions for each channel.
- Histograms and trend graphs of ACD rate counters.

9.4.2.1 ACD Population Maps

The counting frequency of each discriminator channel will be maintained. This will identify both noisy and inefficient channels. The small number of channels will have little impact on memory usage. The anticipated low occupancy of the ACD should result in an equally low impact on CPU resources. Some preselection criteria, such as separating the ACD hits that have a track passing through them from those with no track, could be used to sharpen the distributions.

9.4.2.2 ACD Pulse Height Distributions

Pulse height distributions will be maintained for each of the ACD's 178 channels. The number of histogram bins is determined by the range of the ADC and by the desired resolution. Using the current design, the ACD pulse height is digitized into one of two, twelve bit ranges. Normally the ACD pulse height information is subjected to a threshold cut. Special triggers will be interleaved with normal data taking which disable the threshold cut so that pedestals may be monitored. The ACD currently does not support both ranges of the pulse height information to be taken simultaneously, making it impossible to monitor the overlap region of the two gains during normal running.

The space needed to hold these histograms is manageable. The basic unit is 178 channels * 2^{13} bins per channel = 2 million bins. One must multiply this by the bin width (4 bytes) and any slicing, such as by whether it appears a track passes through the tile or not. An upper limit on ACD monitoring is ~16Mbytes. If this is deemed too costly, then the histograms may be separated into those that monitor pedestals and those that monitor pulse height. The former would have fine-grain bins that cover a limited range, while the latter would have coarser bins that cover the full range. Fine grain coverage around the minimum ionizing range could also be useful. Two sets would be kept; one when a charged track appeared to extend into the ACD tile and one when it did not. The resulting distributions would serve to establish the minimum ionizing pulse height value.

9.4.2.3 ACD Rate Counters

The ACD tiles can also be monitored by a limited set of rate counters. Individual ACD tiles may be multiplexed to these rate counters. After a start signal is given, the counters will increment for a fixed duration. The counters may be used for both science and monitoring reasons. From a monitoring standpoint, they would serve the same purpose as the discriminator population distributions, substituting higher statistics for less selectivity. If it proves useful, the DAQ will also maintain trend graphs of these counters.

9.4.3 ACD Calibration

9.4.3.1 ACD Calibration Procedure

The front-end electronics of the ACD can be calibrated by injecting known signals. This is a classic calibration scheme. A high-precision DAC controls the charge injection. The detector is triggered a number of times with the DAQ system gathering and examining the events at each DAC setting, extracting a mean and a variance for each channel at each DAC value. After all values are gathered, a response curve is fit to the data. This calibrates not only the electronic gain of the ACD, but also serves to tie the 2 gain ranges together.

9.4.3.2 ACD Calibration Background Contamination

One possible complication is the contamination of the calibration data with cosmics. While this problem can be avoided by turning the high voltage to the phototubes off, procedurally and technically this may be difficult or undesirable.

The expected ACD singles counting rate is 1KHz per tile, with an ~3-10usec shaping/memory time. This leads to between .3-1% of the calibration triggers also containing a background track or some remnant of it. It may be possible to lean on the TKR or CAL to provide some help in identifying background contamination, but a robust system design would avoid this technique, making the ACD stand on its own. It may be better to attempt to remove outliers, verifying that the number being removed is consistent with expectations.

9.4.3.3 ACD Calibration, Time and Memory Usage

The time to calibrate the ACD depends on the number of DAC settings and the number points taken at each setting. In calibration mode the zero suppression feature of the ACD DAQ is disabled. The event size is ~1 kBytes (178 channels * 4 bytes). With only 178 channels, the speed of an ACD calibration will be limited by recovery time of the charge injection circuit. CPU usage and memory needed to contain the intermediary results are small. Assuming 1msec recovery time and 100 points at each of 100 DAC values, it will take 10 seconds to calibrate the ACD.

9.5 CAL

9.5.1 CAL Monitoring

During normal running, pulse height information is available from the calorimeter. The large number of channels (~3000) and great dynamic range (2^{14}) prohibit histogramming

every channel at full resolution.

CAL monitoring during normal running will come from sending a sample of CNO events, augmented with protons and Heliums to the ground. The CNO events provide an absolute scale for the calorimeter. The only impact should be the CPU resources used to identify the candidates (~10Hz) and the output bandwidth consumed by the selected events. The latter is the most important, but can easily be controlled by either screening or prescaling the events.

9.5.2 CAL Calibration

9.5.2.1 CAL Calibration Procedure

The front-end electronics of the calorimeter can be calibrated by injecting known signals. This is a classic calibration scheme. A high-precision DAC controls the charge injection. The detector is triggered a number of times with the DAQ system gathering and examining the events at each DAC setting. Means and variances for each channel at each DAQ setting are accumulated and a response curve fit. This calibrates not only the electronic gain of the calorimeter, but also serves to tie the 4 gain ranges together.

9.5.2.2 CAL Calibration Background Contamination

One possible complication is the contamination of the calibration data with cosmics. While this problem can be avoided by turning the bias voltage to the PIN diodes off, the increased capacitance will cause noise problems.

If one assumes a 3usec shaping time and a rate of 10 kHz then approximately 3% of the calibration triggers will also have an overlapping cosmic ray. (10 usecs may be a more realistic time, raising the rate to 10%). This background should be uniformly spread over one layer (192-384 logs, assuming 1-2 logs occupancy per track), reducing the effect to less than 0.1%.

9.5.2.3 CAL Calibration Time and Memory Usage

The time to calibrate the CAL depends on the number of DAC settings and the number of points taken at each setting. In calibration mode both the zero suppression and auto-range features of the CAL DAQ are disabled, resulting in very large events, on the order of 25 kBytes (1.5k logs * 4 ranges * 4 bytes/log). The speed of the calibration will be limited by:

- The bandwidth between the towers and the CPU.
- CPU processing time.

If the bandwidth between the towers and a CPU is 160 Mbit/sec, it will take ~1.25 msecs to transport the event, limiting the pulsing rate to ~800 Hz. Assuming the bulk of processing consists of computing the means and variances of the various DAQ settings, it should take only about .5 usecs per point to process the 25k points, slightly faster than the transport time. This load, however, can be distributed over the all the CPUs. Thus, the limiting factor is likely the bandwidth. If 100 samples are taken at each of 100 points, it will take approximately 10 seconds to calibrate the CAL.

Memory usage is well within reason. Besides the input buffering, one needs to maintain 8 bytes (mean and sigma) for each of the calibration DAC settings per channel or roughly 8

bytes * 100 DAC settings * 3k channels = 2.5 MBytes.

9.6 TKR

9.6.1 TKR Monitoring

Population histograms will be maintained for the tracker during normal running. These will be used to monitor inefficient and noisy strips.

9.6.1.1 TKR CPU and Memory Usage

Each monitored strip costs 4 bytes. With approximately 885k strips, 3.5 MBytes are needed. This will double if hits on tracks are separated from all hits.

CPU usage is significant. This is particularly true if the filtering algorithm does not unpack the tracker data. The burden would then fall to the monitoring services. Estimates are that it will take ~100-500 nsecs/hit to histogram a hit. With 50-100hits/event this will take 10-50 usecs/event, approximately 10-20% of one CPU. If the CPU time is too onerous, the number of events can be prescaled.

9.6.1.2 TKR Rate Counters

Each tracker tower will have at least two rate counters. A multiplexer can be set so that any layer, or more precisely layer end, can be monitored. This may prove to be a more effective strategy in monitoring noisy channels than relying on histograms gathered from events. Abnormally high rates in these counters can be used to identify potential trouble spots. Any such layers are then scrutinized on a strip-by-strip basis.

9.6.2 TKR Calibration

9.6.2.1 TKR Calibration Procedure

The goal of a tracker calibration is to set the discriminators commensurate with a minimum ionizing particle. The Tracker can only be calibrated indirectly. The procedure injects charge into the front-end electronics and detects the result by observing whether the injected charge produces an output signal. This determines the electronics gain of the system. The threshold curve will be mapped by sweeping the value of the injected charge. If the output threshold DAC is also swept, a pulse height distribution can be built one value at a time by effectively translating an input DAC value to an output DAC value.

9.6.2.2 TKR Calibration Background Contamination

If the bias voltage of the tracker is not removed, the tracker is exposed to a rate of approximately 10 kHz of charge particles. This rate, together with a 3 usec shaping time will lead to a background rate of 3-10%. Because this background is spread over approximately one layer (1536 strips * 16 towers) and because the requirements for the tracker are not nearly as stringent as for the ACD or CAL, this background will only be a nuisance, producing a small and uniform offset of noise for each DAC setting.

9.6.2.3 TKR Calibration Time and Resources

A full sweep to determine the threshold curve of the TKR will be fairly expensive. Approximately 100 points at each DAC setting will be required. The range of the DAC can be limited to only those values near the threshold; worst case would be the 2^7 settings available to the charge injection DAC. The real problem is that the DAQ system will limit the number of strips that can be simultaneously calibrated. A layer end readout is limited to maximum of ~128 strips. In addition, the DAQ transport may limit the maximum hits per cable to ~512. It will take 12-15 passes to complete these curves.

Although it is possible to do all towers in parallel, it is unlikely that the CPUs can process all the data at this rate. At 100-500 nsec for each strip hit, it will take each CPU 6-16 msec to process the 64k input values. With four CPUs working together, the aggregate rate is 250-600 Hz. For the 10k points and 15 passes, this will take between 250 and 600 seconds. (A factor of two better if one assumes that 50% of the time will be spent on DAC values below threshold.)

9.7 Global Triggering

9.7.1 GLT Monitoring

While the GLT has no calibration procedure, it will be monitored. The goal is to prove that the output of the GLT is consistent with its inputs. This goal may be achieved in two ways.

During normal data taking, the output of the GLT can be shown to be consistent with the inputs. For the tracker, assuming that the data masks of the TKR are a super-set of the trigger masks, this can be done at the lowest level. The procedure would be to reconstruct the layer masks and the 3-in-a-row masks from the data for each tower.

This procedure is not available to the CAL, because the discriminators that produce the triggers are not recorded in the data stream; at most the ORs of all the tower discriminators are available in the GLT data. They may be inferred from the pulse height values, but this is tenuous. The signal used for the pulse height is not the same of the discriminator, so the correlation of pulse height to discriminator firing is only inferential. This is not good enough if the goal is logical verification.

The ACD is somewhat in between the CAL and TKR. The LO discriminator values are recorded in the data stream, so they can be checked completely. Other ACD trigger signals are not recorded in the ACD data so must be inferred from other information, such as pulse heights.

A more thorough procedure is to control the exact inputs to the trigger by using the charge injection signals to pulse precise patterns. This will verify the logic paths but does not determine the efficiency of some of the paths because of the loose correlation between input signal and output discriminator firing.

10 Transiting the SAA

The LAT instrument will be warned on entry into the SAA by at least two mechanisms. The first line of defense will be from stored command sequences that carry the SAA entry and

exit times. These command sequences will be updated regularly from the ground. It is also anticipated the LAT will carry a robust stand-alone detector dedicated to monitoring SAA entry and exit. The LAT dataflow system will make the counting rate of this detector available to the LAT flight software and the LAT flight software will use this as a back-up system to detect SAA entry and exit. A final line of defense will come from monitoring the currents and/or rate counters of the ACD. Note that this method is only good for detecting entry into the SAA, since, after entry, the high voltage to the ACD phototubes will be off.

11 Ancillary Science Processing

Maintenance of sky maps and identifying transients require that the flight software to convert gamma ray events in LAT coordinates to galactic coordinates. This can only be done with auxiliary information coming from the spacecraft.

11.1 Spacecraft Communication

The spacecraft will provide position, attitude and timing information to the LAT. This information will be passed over the 1553 bus to the SIU along with a GPS "time hack" on a discrete signal line. The SIU will use the information from the spacecraft to determine the location of the Earth's limb in terms of the LAT's coordinates. The attitude information will be updated at a rate of 5 Hz. A GPS timing signal is received at 1 Hz.

11.2 Detecting Transient Events

The LAT will maintain an on-board sky map. This map will be updated with all probable photon events. The map will be maintained to TBD level of precision. When the gamma ray detection rate from a particular direction exceeds TBD Hz, the LAT will use the on-board map to identify the transient source.

11.2.1 LAT - GBM Interaction

The LAT interacts with the Gamma Ray Burst Monitor in two ways.

- GBM alert messages
- GBM 1553 messages, three types
    - Rapid Burst with Location
    - Burst Downlink Information
    - Repoint Candidate Message or Burst Over Message

The GBM alert represents prompt notification (within 5msecs) by the GBM of the *possibility* of a GRB event. The message is carried by a dedicate wire. The LAT will see this signal as an interrupt message. No information, other than its presence, is carried by the GBM alert. The LAT's response will be to loosen the filtering requirements, allowing more candidate events to be gathered.

Within 2 seconds of receiving this interrupt, the GBM will issue a Rapid Burst with Location message, received over the 1553 bus, detailing the exact nature of the event.

The GBM will also issue update messages, in the form of Burst Downlink Message, during the coarse of the GBM. These are primarily meant for ground use, so the LAT flight software will likely ignore them.

If the event is sufficiently interesting, the GBM may request the LAT to issue a repointing request on its behalf using a Repoint Candidate Message. Alternatively at this point, a Burst Over Message from the GBM indicates that the event is no longer interesting. In the latter case, the LAT can resume its nominal operation.

11.3 Requesting Repositioning Upon Transient Event Detection

For transient event candidates, the LAT will take attitude information from the spacecraft and determine the time that the source will remain within the LAT's effective field of view. If this time is greater than TBD seconds and the LAT flight software determines the source is sufficiently interesting, the LAT will request the spacecraft to be pointed at the transient. Note that this is only a *request* by the LAT to the spacecraft. The LAT flight software must gracefully handle a refusal by the spacecraft to repoint. As noted above, the GBM may also issue a repointing request. This request is always passed through the LAT, before being passed on to the spacecraft. The LAT, itself, may choose not to pass the request on.

11.4 Ground Event Notification

If the LAT observes a transient from a region that is not already identified on the on-board sky map with a flux above TBD, the LAT will initiate real time ground notification through TDRSS multi-access. This notification will include the position of the transient, the time of the leading edge of the transient, and the current or peak flux (whichever is greater) of the transient.

12 System Margin

Per- DOD-HDBK-343, for class A or B missions, the system margins at PDR/CDR should be at least

- A factor of 2 in processor power.
- 1.5 times the expected memory needs.

As a design goal, flight software should be designed and developed to preserve, as margin at launch, 30% of RAM capability and 25% of the CPU processing capability of the command processor and the orbit average capability of the data processors. Processing margin determination shall include processor diagnostics.

13 Software Testing

This section defines the general requirements for test and verification of the GLAST-LAT Flight Computer Software. These requirements assure that the functional and performance requirements are properly met. Formal verification of compliance shall be the basis for acceptance of all flight Computer Software Configuration Items (CSCIs). The LAT shall abide by NASA software quality and assurance guidelines, in particular, the NASA SOFTWARE ASSURANCE GUIDEBOOK, NASA-GB-A201

13.1 Qualification Process

The qualification process consists of

- Informal multi-level software testing, as described in the following paragraphs,
- The Formal Qualification Test (FQT) utilizes the Flight Software Validation Test Procedure performed and witnessed in accordance with the LAT's Software Management Plan, LAT-MD-104.1.

The multi-level software test structure shall be used in the software qualification process.

Table 8 provides an example of the qualification methods used to verify the individual capabilities and subcapabilities for a sample of CSCIs. Here, *Test* refers to formal or informal testing using the actual software. *Anal*ysis means inspecting the code and verifying that the requirements are met. Analysis is particularly important when assessing the real time behavior of the software. Setting up reproducible tests to duplicate and probe real time problems is difficult at best. In practice, both are necessary.

Paragraph	Title	Level 0	Level 1	Level 2
	System Initialization	N/A	Test	Test
	Boot Strap Loader	Test	Anal	Anal
	Boot 1553 Init	Test	Anal	Anal
	EEPROM Handler	Test	Anal	Anal

Table 8 Qualification Methods for Software Capabilities on a Sample of CSCIs

13.1.1 Level 0 Tests

Level 0 testing is performed on the smallest software element that is a separately compilable function. Level 0 testing methods shall be selected to achieve the results in accordance with the following criteria:

Verify error free compilation and unit code integrity. Verify adherence to the specified programming standards of the GLAST-LAT SDP.

- Verify proper function execution to include control flow, logic paths, error detection, and error recovery
- Verify compliance with processing accuracy requirements.
- Verify compliance with memory and timing requirements.
- Verify ability to accept and properly process the complete design range of input data and parameters.
- Verify output functions and associated formats.

Each individual software element, starting with the top of the software component structure, shall be complied to produce an error free listing and a binary file. The integrity of the

software element is determined by manually verifying that the compilation listing fulfills the applicable requirements stated in the Software Requirements Specification. Next an executable program is generated using "dummy" software elements where necessary to enable closed loop execution of the software element under test. The manufacturer supplied "debugger" shall be utilized as required to setup inputs, display results, verify logic paths, and troubleshoot any functional errors. If software errors are identified, the software element shall be revised and retested until agreement is reached between the requirements and the software execution. Results of this testing shall be documented in the LAT Software Development File in accordance with the provisions of the Software Development Plan.

13.1.2 Level 1 Tests

Level 1 tests shall be performed on an integrated group or string of related software elements. Each individual software element shall have successfully completed Level 0 testing. Level 1 test methods shall be selected to achieve results in accordance with the following criteria:

- Verify that the intercomponent communication functions transfer control and data/ parameters correctly, and transactions are within allowable timing constraints.
- Reaffirm the mathematical accuracy and processing abilities of the software elements.
- Verify the software elements operational environment interfaces where applicable.

Level 1 testing shall utilize preplanned test cases, employing the same techniques as Level 0 testing of software elements. The testing shall include incremental integration of software elements as each component is verified. As related software elements complete verification, the integration testing of these related software elements to form a CSCI shall be performed. Such testing shall verify the ability of the component to perform assigned functions correctly when executed within the framework of other operational components. Incremental integration shall begin with one component and proceed to encompass the entire collection of components. Even though a collection of components is involved, the testing shall concentrate on the exercise of the single component being tested and its interfaces with other operational components. Test cases and test results shall be documented in the LAT Software Development File in accordance with the provisions of the Software Development Plan.

13.1.3 Level 2 Tests

A CSCI shall be defined to be an integrated set of software components that represent a complete software system. Such systems shall have successfully completed Level 0 and Level 1 testing. Testing at Level 2 shall include exercising the software using real data in the actual operational environment to verify its ability to perform the required functions in both nominal and non-nominal modes. Level 2 testing methods shall be selected to achieve results in accordance with the following criteria:

- Reaffirm the ability of the software to reproduce the Level 0 and Level 1 test results.
- Verify software element and software environment interfaces.

- Verify compliance with the requirements for system level functions, timing relationships, and processing accuracy's.
- Verify closed loop software operation.

One of the critical tasks of Level 2 testing is software/hardware integration. The first step is to verify the individual software elements that interface directly to the external environment (I/O drivers) operate properly. After the I/O driver software elements are verified, the functional utilization of the interface in the software system is tested. The final step shall be to execute a typical sequence that utilizes the major capabilities designed into the software system. The results of this test sequence shall be predicted, and compared to the actual results. Successful execution and verification of a match with predicted results shall demonstrate successful completion of "informal" Level 2 testing. Test cases and test results shall be documented in the LAT Software Development File in accordance with the provisions of the Software Development Plan.

13.2 Formal Qualification Testing

The software Formal Qualification Test (FQT) is the buyoff that the flight software meets its formal requirements. Because of the complexity of GLAST this test is not as easy to define as for other programs. The FQT is actually a series of tests that together verify the compliance of the flight software to the requirements as specified in the Software Requirements Document. Those elements of the software that are associated with functions that have not only well defined interfaces but predictable execution paths can be tested using traditional methods of verifying correct response to stimulus. These elements include 1553 Bus and other vehicle interface software and instrument command, control, and housekeeping. The other software elements, that are associated with science analysis tasks such as event filtering and transient event detection, will most likely require more statistical types of testing. The first step in testing will be to use specific test events to verify consistent response to a given stimulus. But beyond this it will be necessary to utilize libraries of events that are a mixture of photons and background events and which are different from the event libraries used for software development. This is necessary to avoid the tailoring of the software to the recognition of specific events. The event libraries should faithfully duplicate the expected flight environment in both spectral and spatial distribution to allow the background rejection and processing efficiencies to be determined.

FQT modules:

1. Vehicle command interface
2. Vehicle telemetry interface
3. 1553 Bus telemetry
4. Wideband telemetry
5. Instrument command queues
6. Instrument housekeeping
7. Instrument internal command and configuration
8. Event filtering

9. Additional science processing
10. Instrument calibration

14 Schedule/Manpower

A rolled up version of the WBS is presented below. The Flight Software Schedule revolves around four major milestones. These milestones correspond to the delivery of the Engineering Model 1 and Model 2 hardware, the first Qualification Unit and the final Flight Units.

14.1 Engineering Model 1

Engineering Model 1 serves as the first realistic hardware platform for the LAT DAQ and FSW. EM1 facilitates the development of all software needed for a single CPU to interact with a single TEM. Many of the low level utilities that will form the basis of the system are developed using EM1 as a target. Software needed to command and configure as well as pull data from a single tower will be written and tested. Software to read the housekeeping and Low Rate Science data will be implemented with EM1. A framework to support the flight software can be prototyped.

In addition to providing a software development environment, EM1 will also provide a test bench for exercising and testing front-end electronics. FSW will help in the development of software needed for these activities.

A hardware simulator will be developed at this time. The goal of the hardware simulator is to act as a surrogate TEM, pushing data into the DAQ system. The hardware simulator tests the DAQ dataflow paths, providing a means to measure performance and spot potential problems. Software to use and exploit the hardware simulator needs to be designed and implemented during this period.

14.2 Engineering Model 2

Engineering Model 2 builds from EM1, adding multi-CPU and multi-TEM capability. Scatter/gather versions of software used in EM1 will be needed. For instance, statistics that were accumulated in EM1 on one CPU now will be accumulated on at least two CPUs. The results from both CPUs must be gathered and combined.

A real commanding environment will take shape in EM2. EM2 should provide support for GSE activities. The calibration procedures are designed, written and tested.

Since EM2 has all the features of the actual instrument, it serves as the main development platform for the FSW. All that it is missing is a full complement of CPUs and TEMs. These will gradually be added as time passes, building up to at least 2 CPUs and 4 TEMs.

14.3 Qualification Unit

The Qualification Unit represents a fully operational version of one TEM. All hardwire revisions due to problems found during EM1 and EM2 testing should be in place. Software patches that attempted to sidestep these problems will be removed and the Qualification Unit exercised.

The emphasis here is not on developing more software, but on solidifying it. Functionality will be added, but activity will be tilted more to exploiting existing facilities than on developing new ones. System programming takes a backseat to application programming.

14.4 Flight Units

The Flight Units' arrival signal the beginning the integration and testing phase of the instrument. All CPUs and electronics modules can be assembled. Many, but likely not all, real life operational issues and scenarios can be exercised.

14.5 Test Bench Support

As LAT integration proceeds, there will come a point when subsystem written test bench software will reach a sufficient level of sophistication that it can form the basis of a flyable instrument diagnostic suite. It is in flight software's interest to become involved in diagnostic software development at this point. In exchange for supporting subsystem software efforts, flight software gains valuable insight into the idiosyncrasies of the LAT detectors "from the horse's mouth". Such a course also avoids duplication of effort and helps achieve the "test it as you fly it" goal.

14.6 Manpower

The Flight Software Group consists of ~6 people. SLAC will act as the center of operations with 4 full time people. NRL will contribute 2 people, 1 exclusively devoted to LAT FSW. Another person at Texas A&M at Kingsville acts as a liaison between Flight Software and ground software, providing Monte Carlo data that is used in various studies of the DAQ and Flight Software.

14.7 WBS

WBS	Task Name	Duration	Start
4.1.7.9.1	**Infrastructure Development/Test**	**1207 days**	**5/1/01**
4.1.7.9.1.1	Infrastructure Development	1207 days	5/1/01
4.1.7.9.1.2	Test Bench Support	1207 days	5/1/01
4.1.7.9.2	Determine CPU Resources	116 days	5/1/01
4.1.7.9.3	**Engineering Model 1**	**337 days**	**8/31/01**
4.1.7.9.3.1	Architecture	337 days	8/31/01
4.1.7.9.3.2	Low Level Diagnostics	69 days	5/10/02
4.1.7.9.3.3	Tower Command & Configuration	150 days	11/19/01
4.1.7.9.3.4	Dataflow	150 days	11/19/01
4.1.7.9.3.5	Housekeeping	150 days	11/19/01
4.1.7.9.3.6	EM1 Simulator	40 days	10/26/01
4.1.7.9.3.7	Low Rate Science	24 days	11/19/01
4.1.7.9.4	**Engineering Model 2**	**192 days**	**6/26/02**
4.1.7.9.4.1	EM1/EM2 Transition	16 days	6/26/02
4.1.7.9.4.2	Low Level Diagnostics	158 days	6/26/02
4.1.7.9.4.3	Facilities	158 days	6/26/02
4.1.7.9.4.4	Fast Monitoring	158 days	6/26/02
4.1.7.9.4.5	Space Craft OPs	67 days	11/8/02
4.1.7.9.4.6	GSE	7 days	1/31/03
4.1.7.9.4.7	Uplink/Downlink	7 days	1/31/03
4.1.7.9.4.8	Calibration	158 days	6/26/02
4.1.7.9.4.9	EM2 Simulator	158 days	6/26/02
4.1.7.9.4.10	Science	35 days	6/26/02
4.1.7.9.4.11	**Multi-CPU Tool Box Effort**	**192 days**	**6/26/02**
4.1.7.9.4.11.1	Multi-CPU Tool Box	192 days	6/26/02
4.1.7.9.4.11.2	Multi-CPU Low Level Diagnostic	158 days	6/26/02
4.1.7.9.4.11.3	Multi-CPU Calibration	192 days	6/26/02
4.1.7.9.5	**Qualification Unit**	**55 days**	**6/11/03**
4.1.7.9.5.2	**Multi-CPU Implementations**	**55 days**	**6/11/03**
4.1.7.9.5.2.1	Facilities	55 days	6/11/03
4.1.7.9.5.2.2	Fast Monitoring	55 days	6/11/03
4.1.7.9.5.2.3	Space Craft Ops	55 days	6/11/03
4.1.7.9.5.2.5	GSE	55 days	6/11/03
4.1.7.9.5.2.7	Science	55 days	6/11/03
4.1.7.9.6	**Flight Unit**	**711 days?**	**8/27/03**
4.1.7.9.6.1	Final Instrument Code Integration	111 days	8/27/03
4.1.7.9.6.2	SLAC Based Flight I&T	315 days	5/12/04
4.1.7.9.6.3	Non-SLAC Base Flight I&T	211 days?	7/27/05

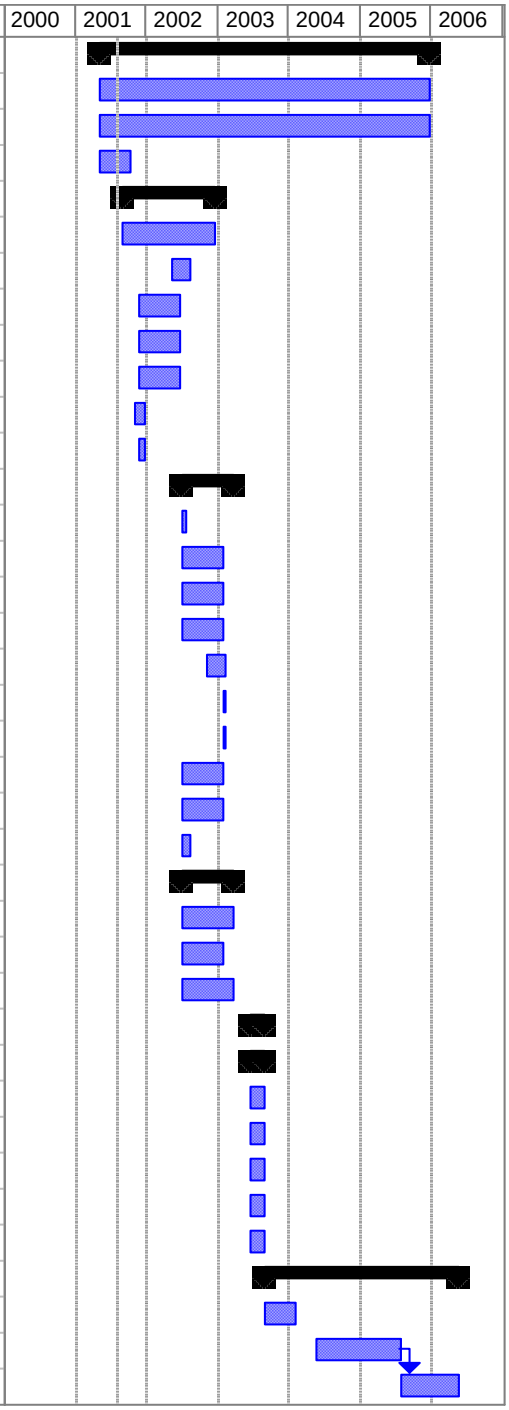

Table 9 Rolled up WBS