Flight Software Global Design Specification

James Francis

MIT 64-30102

Rev. G

November 14, 1994

Contents

Figures

1 Introduction

1.1 Purpose

The purpose of the Flight Software Global Design Specification is to describe the global architecture and design of the Experiment Data System (EDS) Flight Software on the Xray Timing Explorer (XTE). It describes the interfaces and defines the modules which make up the EDS Flight Software. It is intended to be used as a superior level document in the development of the module designs.

1.2 Scope

The EDS Flight Software is responsible for acquiring and processing data from the Proportional Counter Array (PCA) and the All-Sky Monitor (ASM), transferring the processed data to the Spacecraft's Flight Data System (FDS), which telemeters the information to the ground, and for managing the rotation of the ASM cameras. All software operations are ground configurable. This allows the experiment to process the measured data in a highly flexible fashion. The processed data can then be transferred to the ground using limited telemetry resources.

1.3 Definitions

Xray Timing Explorer (XTE). The Xray Timing Explorer consists of a Spacecraft carrying three astronomical Xray experiments.

Flight Data System (FDS). The Flight Data System contains the main Spacecraft computer which handles all interfaces to the ground.

Experiment Data System (EDS). The EDS consists of 8 event analyzers and some interface hardware, built by MIT.

All-Sky Monitor (ASM). The ASM consists of three Scanning Shadow Cameras (SSC's) and a rotation assembly, built by MIT.

Proportional Counter Array (PCA). The PCA consists of 5 Proportional Counter Units (PCU), built by NASA/GSFC.

High Energy Xray Timing Experiment (HEXTE). HEXTE is a high energy timing experiment, built by the University of California, San Diego (UCSD).

HEXTE has the capability of detecting xray bursts and triggering the EDS. The EDS can also detect interesting xray bursts and trigger HEXTE.

Event Analyzer (EA). Each Event Analyzer on the EDS is responsible for processing events from either the PCU's or from the SSC's.

Consultative Committee for Space Data Systems (CCSDS). The FDS sends and receives CCSDS conformant command and telemetry packets to and from the EDS.

Big-endian / Little-endian. These terms refer to bit or byte ordering conventions used by command and telemetry packet format descriptions and Flight Software memory layout definitions. Big-endian refers to a convention where data is represented most-significant bit or byte first. Little-endian refers to a convention where data is represented least-significant bit or byte first.

1.4 References

[1] MIT 64-02009 Rev. E .... "EDS-PCA Science Requirements"
[2] MIT 64-02012 Rev. C .... "EDS-ASM Science Requirements"
[3] MIT 64-02006.01 Rev. E .... "XTE EDS Hardware Functional Description and Requirements"
[4] MIT 64-03006 Rev. C .... "ASM/EDS Interface Control Document"
[5] MIT 64-03007 Rev. D .... "FDS to EDS Interface Control Document"
[6] MIT 64-03001 Rev. I .... "PCA/EDS Interface Control Document"
[7] MIT 64-03005 Rev. B .... "HEXTE/EDS Interface Control Document"

1.5 List of Acronyms and Abbreviations

ASM .... All Sky Monitor
BREGS .... Backend Registers
CCSDS .... Consultative Committee for Space Data Systems
CPU .... Central Processing Unit
DMA .... Direct Memory Access
DSP .... Digital Signal Processor
EA .... Event Analyzer
ECC .... Error Correction Code
EDS .... Experiment Data System
EEPROM ..... Electronically Erasable Programmable Read-only Memory
FIFO .... First-in, First-out (usually refers to memory)
FDS .... Flight Data System
FPGA .... Field Programmable Gate Array
FREGS .... Frontend Registers
FS .... Flight Software
GSFC .... Goddard Space Flight Center

HEXTE .... High Energy X-Ray Timing Experiment
HW............... Hardware
ICD .... Interface Control Document
I/O .... Input-Output
LED .... Light Emitting Diode
LSB .... Least Significant Bit
LUT .... Look-up Table
MET............. Mission Elapsed Time
MSB .... Most Significant Bit
NASA .... National Aeronautics and Space Administration
NULL........... A value equal to zero.
PRAM .... Protected RAM
PCA .... Proportional Counter Array
PCU .... Proportional Counter Unit
PSDU .... Power Switching and Distribution Unit
RAM .... Random access memory (read-write memory)
ROM .... Read-only memory
RT .... Remote Terminal
S/C .... Spacecraft
SEU.............. Single Event Upset
SSC .... Scanning Shadow Camera
SW .... Software
TBD .... To be determined
TBR .... To be reviewed
TBS .... To be specified
UTC .... Universal Time (Coordinated)
XTE .... X-Ray Timing Explorer

1.6 Overview

Sections 1 - 2 provide a general description of this document and the Flight Software, within the context of the Xray Timing Explorer. Section 3 describes the major interfaces to the Flight Software. Section 4 describes the overall design of the Flight Software. Section 5 lists the resources needed to test the Flight Software. Section 6 addresses requirement traceability issues, and Section 7 describes how the Flight Software is packaged.

2 General Description

2.1 Overview

This section lists and briefly describes the major components of the Xray Timing Explorer.

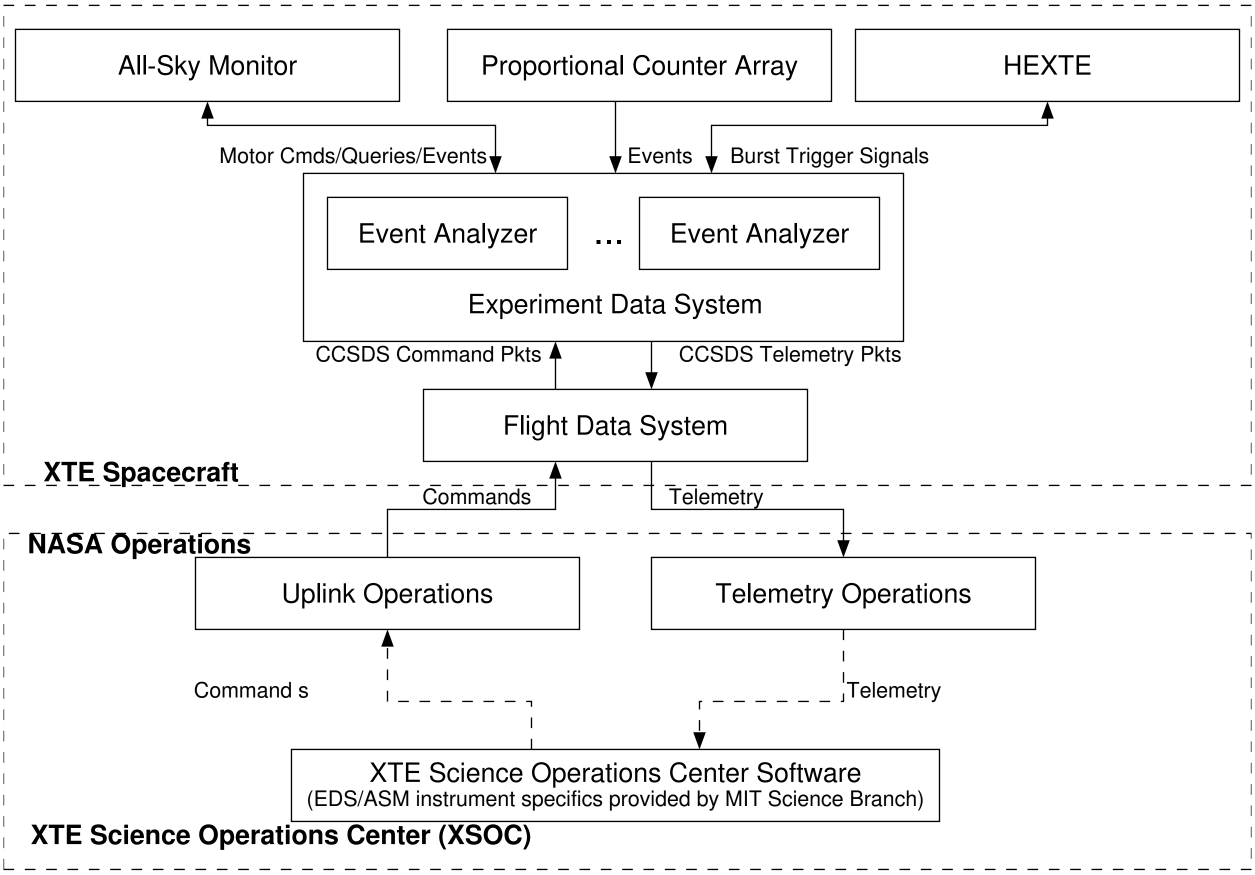


Figure 1 XTE and Ground Systems Block Diagram

The following lists each major component and briefly describes the purpose of each:

XSOC Software .... The XTE Science Operations Center Software contains portions of software which are dedicated to supporting the XTE Experiment Data System and All-Sky-Monitor. These software items are provided by the MIT Science Branch to the SOC. The primary functions of the provided software include generating commands for the Flight Software in each EA and, using telemetry packets generated by the Flight Software, monitoring the operation of the EDS and ASM. These software items are also responsible for unpacking the science data for each specific observation.

Uplink Operations............................ NASA is responsible for controlling the XTE Spacecraft, and for accomplishing the physical uplink of commands.

Telemetry Operations .... NASA is responsible for receiving and archiving telemetered data from the Spacecraft (including that from the experiments).

Flight Data System (FDS) .... The Flight Data System is responsible for receiving commands from the ground, and for storing and delivering these commands to the EDS. It is also responsible for receiving telemetry data from the EDS and telemetering these data to the ground.

Event Analyzers (EA)....................... The EDS contains 8 Event Analyzers. The Event Analyzers and the Flight Software will be responsible for acquiring and processing data from the PCA and ASM detectors.

All-Sky Monitor (ASM) .... The All-Sky Monitor consists of 3 Xray cameras mounted on a rotation mechanism. The purpose of these cameras is to perform a survey of the sky for Xray sources. The Flight Software will be responsible for controlling the camera rotations, and for acquiring and processing the data from the cameras.

Proportional Counter Array (PCA)... The Proportional Counter Array consists of 5 Xray detectors. The purpose of these detectors is to measure Xrays from known sources. The Flight Software will be responsible for acquiring and processing the data from the detectors.

HEXTE .... The purpose of HEXTE is to measure high energy Xrays from known sources. The EDS Flight Software will be responsible for triggering HEXTE when an "interesting" set of data is detected in the PCA data, and for capturing high resolution PCA data when triggered by HEXTE.

The following illustrates the major components of the EDS Event Analyzer hardware:

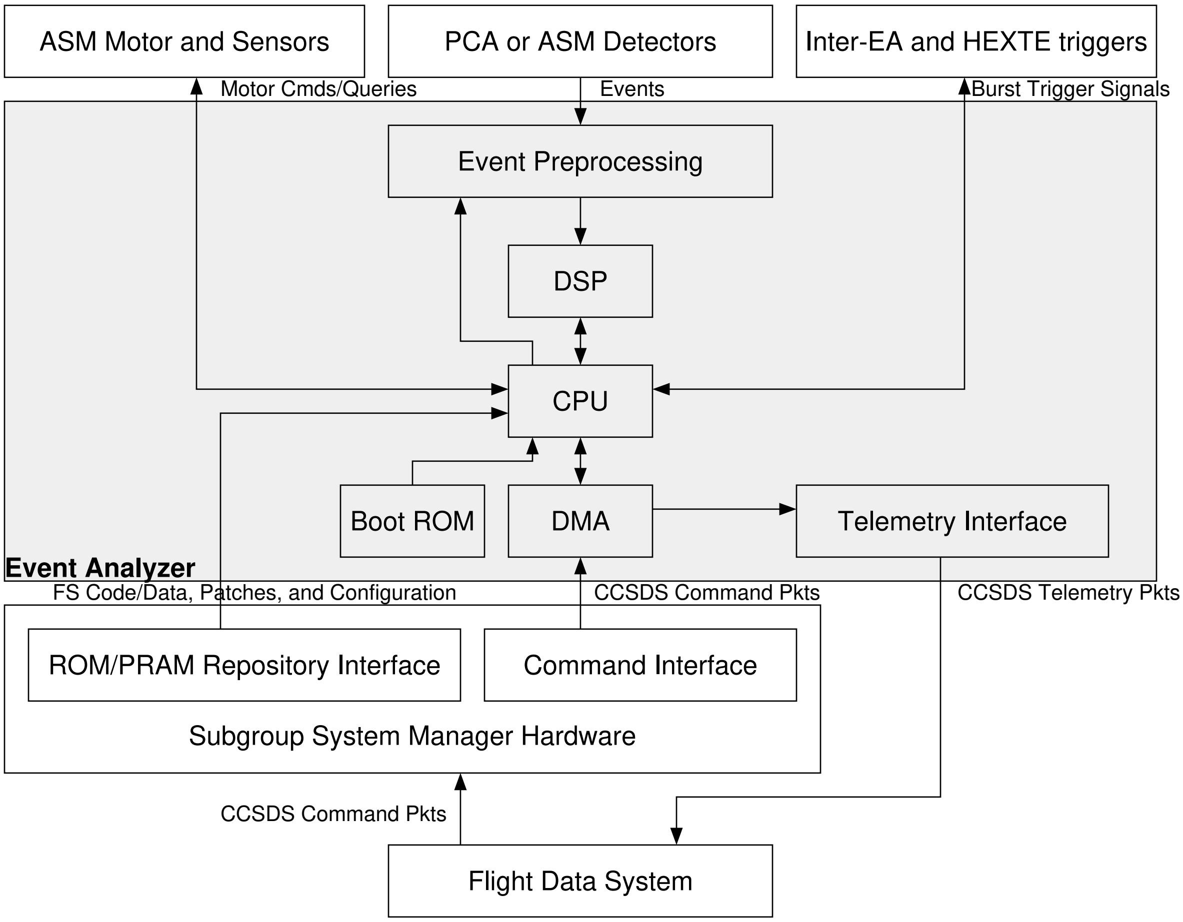


Figure 2 Event Analyzer Block Diagram

The Flight Software consists of two parts. One part runs on the EA's main CPU. This software is responsible for executing commands, controlling the experiment, and for sending the processed data to the FDS. The second part runs on the EA's DSP. This part is responsible for the bulk of the event processing.

The following diagram illustrates the system data flow, with respect to the XSOC, the Spacecraft and the EDS:

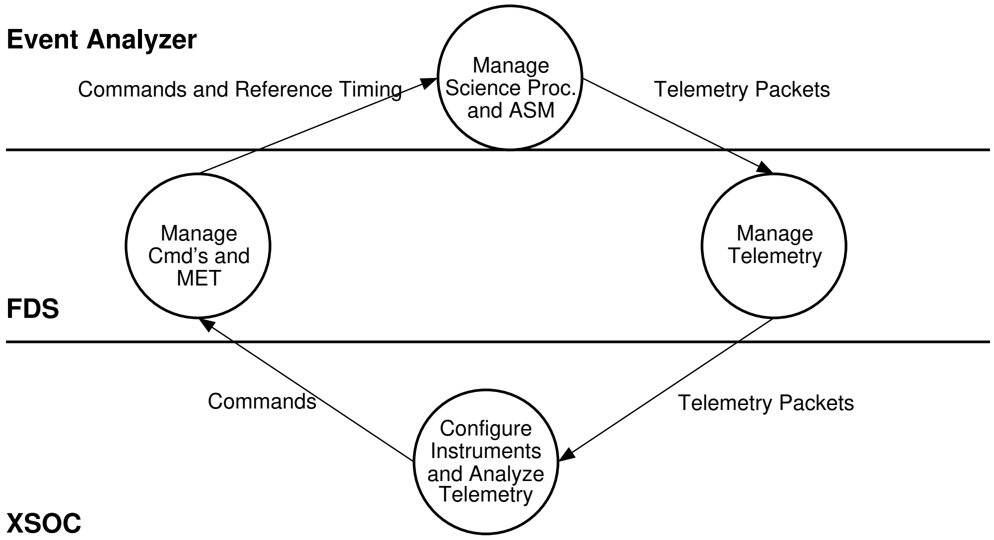


Figure 3 XSOC, FDS, and EDS Data Flow

The following illustrates the life-cycle of a typical science observation, running on one of the the Event Analyzers, where bolded text reflects actions initiated by the ground:

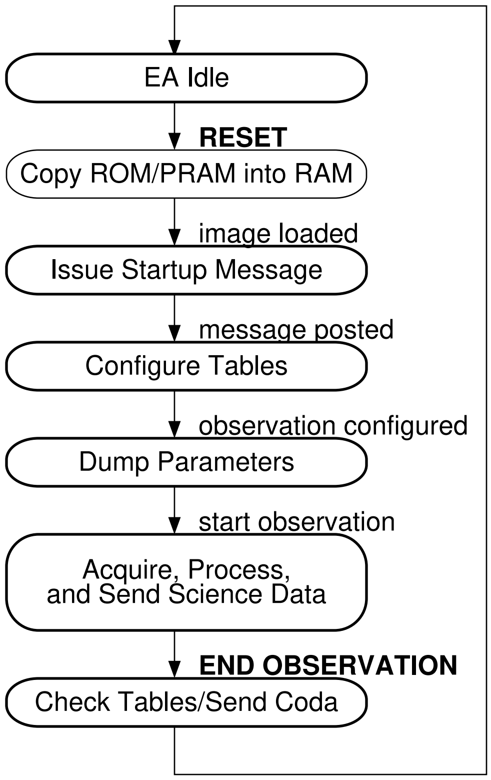


Figure 4 Science Observation Life-Cycle

The ground starts by loading configuration parameters into the System Manager's Protected RAM (PRAM) (not shown). It then issues a reset command to the Event Analyzer. The EA boot procedure copies the ROM and PRAM into its internal memory. It then issues a Startup Message, to indicate to the ground that the EA booted succesfully. It uses the parameters from the copied ROM or PRAM to configure the observation's tables and run-time variables. Once the observation is configured, the EA dumps a copy of all of the observation's parameters to the ground. After the parameters have been dumped, the EA schedules the start of the observation. Once the observation starts, the EA acquires, processes and telemeters science data. To end the observation, the ground sends a command to terminate the observation. When the EA receives this command, it stops data processing, checks the integrity of the observation's tables, and telemeters the results of these integrity checks. It then enters an idle state, responding only to commands, until the ground issues another reset. In the event of a crash or panic during an observation (not shown), the EA will automatically reset, reload ROM and PRAM, and restart the observation. Since the Flight Software cannot write to the PRAM on the System Manager Board, and the PRAM is radiation-hard, it cannot be corrupted by Single-Event upsets (either directly, or by a resulting Flight Software crash).

2.2 Primary Functions

This section lists the major functions provided by the EDS Flight Software.

Configure Observations
Obtain Configuration Info
Setup Lookup Tables
Setup Timing Hardware
Load DSP Code/Data

Manage ASM
Control ASM Motor
Obtain Engineering Data
Transfer Data to FDS

Run Observations
Schedule Start Time
Control Timing Hardware
Process Event Stream
Pack Processed Data
Analyze Processed Data
Transfer Data to FDS

Monitor Health
Touch Watchdog Timer
Acknowledge Commands
Acquire SW Statistics
Provide Obs. Integrity Info
Dump Obs. Parameters

Figure 5 Flight Software Major Functions

2.3 User Characteristics

The Flight Software is intended to be commanded and monitored by parts of the XTE Science Operations Center (XSOC). The developer(s) of the XSOC, and the user(s) of the XSOC, should have detailed knowledge of the operation, architecture and implementation of the Flight Software.

2.4 General Constraints

This section lists the general constraints on the Flight Software:

1) Hardware Performance
The performance capabilities of the Flight Software are limited by the hardware performance, clock rates, timing requirements, etc.

2) Uplink Rate
The rate at which commands can be uplinked to the FDS limits the degree to which the Flight Software can be configured. For example, on the Event Analyzers for the PCA there are 64K bytes of event lookup tables. Since the amount of time it would take to uplink complete literal tables is too long for most observations, the Flight Software must implement an encoding scheme for the tables that allows the user to perform a basic set of operations, using a reasonable amount of uplink.

2.5 Assumptions and Dependencies

The following lists some assumptions made by the Flight Software, and items on which it depends:

1) Telemetry Errors
The Flight Software assumes that the FDS is capable of storing and sending telemetry packets to the ground with no errors.

2) Uplink Errors
The Flight Software assumes that there will be no errors in the received command packets.

3) Number of EA's
The Flight Software assumes that there are a total of 8 Event Analyzers in the EDS.

3 Interfaces

3.1 Event Analyzer Hardware

3.1.1 Overview

The EDS consists of 8 Event Analyzers (EA's), grouped into two subgroups. Each subgroup contains 4 EA's, one ASM-EA, and three PCA-EA's. The subgroup's System Manager hardware supplies each EA with an interface to the ROM and Protected RAM (PRAM) Repository, the FDS's command delivery system, and the FDS supplied Mission Elapsed Time (MET) value.

Each EA consists of two parts, known as the Back End (based on a 80286 microprocessor) and Front End (based on a 320C25 Digital Signal Processor (DSP)) of the EA. The EA's Back End is responsible for command execution, observation control, and telemetry generation. The EA's Front End is responsible for the bulk of the event processing.

For more detail, refer to the **XTE EDS Hardware Functional Description and Requirements Specification**.

3.1.2 Back End

The Back End of an EA consists of the following components:

INTEL 80286 .... This is the EA's main CPU. The portions of the Flight Software responsible for command execution, telemetry generation, observation control, and ASM rotation control run on this processor.

64K bytes Boot ROM .... When an EA is reset, the CPU starts executing from this ROM. The software in the ROM then loads the main FS from either uplink or from the subgroup ROM/PRAM Repository, and transfers control to the loaded code.

Repository Register .... This is a read-only register that the Back End uses to load the Flight Software from the ROM Repository, and patches and configuration information from the PRAM Repository. There are 128K bytes of ROM and 64K bytes of PRAM in the repositories.

256K bytes of RAM .... This is the main CPU memory. The Flight Software running in the Back End and the data reside in this RAM.

8259 Interrupt Controller .... This provides prioritized interrupt control for the 80286. Interrupts are caused by the following:
Interval Interrupt .... Interval timer.
Downlink Interrupt ... Downlink Transfer completed.
Uplink Interrupt .... Uplink Transfer completed.
MinorTick Interrupt .. MinorTick pulse occurred.
First-Flag Interrupt ... DSP read flagged interval marker.

82237 DMA Controller .... The Flight Software uses the DMA controller to obtain command packets from the FDS command interface hardware and to transfer telemetry packets to the FDS. NOTE: The hardware treats command and telemetry packets as a series of 16-bit words. It assumes that these packets are stored little-endian (least-significant byte first) in the EA's memory, but are transferred big-endian (most-significant bit first) across the FDS interface. It does not, however, compensate for data elements larger than one 16-bit word. This forces the Flight Software to use a mixed ordering convention when interpreting or forming portions of the CCSDS headers transferred by the DMA. See Sections 3.2.1 and 3.2.2 for more detail.

MET Register .... The subgroup hardware updates this register whenever the FDS posts a new MET value.

CPU Control Register .... This is a readable/writeable register which controls some Back End hardware operations. These operations include; requesting downlink service from the FDS, and setting the LED states.

CPU Status Register .... This is a read-only register which provides the FS with the status of various Back End components. This information includes; latched versions of the Major and Minor tick clock pulses, latched Uplink and Downlink interrupts, indication that the Watchdog timer is active, the Event Analyzer Id, and the state of the Uplink Boot flag.

CPU Pulse Register .... This is a write-only register which controls some of the Back End functions. These include; resetting of the Major and Minor clock latches, resetting the Uplink and Downlink interrupt latches, and resetting of the Watchdog's counter.

Observation Control Register .... This is a read/write register which the Back End uses to control some of the Front End operations. These include; timing control flags, control of the DSP reset line, and control of the output trigger signals.
NOTE: The DSP reset line also stops interval generation, resets the DSP's Input FIFO, and enables CPU access to the Front End hardware Typer and Recoder Lookup tables.

Observation Status Register............. This is a read-only register which the Back End uses to obtain status information from the Front End. These include; latched versions of the input trigger signals, latched interval and first-flag interrupts, and a latched version of the DSP's Input FIFO spillage flag.

Observation Pulse Register .... This is a write-only register which the Back End uses to control some of the functions of the Front End hardware. These include; resetting of the latched interrupts, and resetting of the latched input triggers.

DSP Memory Interface .... The Back End CPU has read and write access to the Front End DSP's external RAM. This RAM is memory mapped into the 80286's address space.

Front End Lookup Table Access ...... The Back End CPU has read and write access to the PCA Front End hardware's event lookup tables. Although these tables are memory mapped into the 80286's address space, they can only be accessed when the Front End's DSP is held in a reset state. The total size of these tables is 64K bytes.

Interval Counter .... This is a write-only register accessable by the Back End. It controls the duration of the next interval and is specified in microticks.

BinClock Counter .... This is a write-only register accessable by the Back End. It controls the number of microticks per time-bin.

ASM Interface Register .... This read/write register is used to control the ASM motor, to issue ASM engineering sensor queries, and to obtain the results of the queries.

3.1.3 Front End

The Front End of an EA consists of the following components:

TI 32025 DSP .... The DSP is responsible for the bulk of the event processing on an EA. A portion of the Flight Software runs on this processor.

64K words External RAM .... This serves as the DSP's external RAM. The Back End CPU has read and write access to this RAM. 16K words of this space are devoted to program memory, and the remaining 48K words are devoted to data.

Input FIFO .... This is a 1024 16-bit word FIFO which the DSP uses to read event words and Interval Markers from the event processing hardware.

Front End Lookup Tables .... The Front End PCA event processing circuits contain 2 32K byte event lookup tables. The DSP cannot access these tables. It is the Back End's responsibility to configure these tables.

3.2 Packet Formats

This section describes the command and telemetry packet formats visible to the Flight Software (see the **FDS to EDS Interface Control Document**). In addition to these packet formats, the EDS supports discrete command and telemetry packets, and command packets addressed to the PRAM Repository. These packets are received or sent by either the EDS hardware or by the FDS. The formats for these packets are not described in this document.

3.2.1 Command Packets

All command packets, sent by the FDS, start with a CCSDS compliant header, followed by a secondary header, followed by EDS specific information. Command packets have a maximum length of 64 bytes and are always an even number of bytes. The hardware ensures that all packets received by the EA start with the CCSDS header. The FDS guarantees that command packet deliveries are separated at least 31.25ms. Each 16-bit word of the command packet is stored by the DMA in little-endian format (least significant byte first). The following diagram illustrates the format of all commands sent to the EA's:

```
[v,v,v,t,h,a,a,a,a,a,a,a,a,a,a,a] ............ CCSDS Header Word 0
                                               v ............................... Version = 000
                                               t ................................ Type = Telecommand = 1
                                               h ............................... Secondary header present = 1
                                               a ............................... Application Id
[g,g,s,s,s,s,s,s,s,s,s,s,s,s,s,s] .............. CCSDS Header Word 1
                                               g ............................... Segment Flag = None = 11
                                               s ................................ Sequence Number
[z,z,z,z,z,z,z,z,z,z,l,l,l,l,l,l] ................ CCSDS Header Word 2
                                               z ............................... Top 10 bits of packet length = 0
                                               l ................................ Lower 6 bits of packet length
[r,f,f,f,f,f,f,f,c,c,c,c,c,c,c,c] ................ Secondary Header Word 0
                                               r ................................ Reserved
                                               f ................................ Function Code (used by hardware)
                                               c ............................... Packet Checksum
[o,o,o,o,o,t,t,t,r,r,r,r,r,r,r,r] ................. EDS Command Word 0
                                               o ............................... Command Opcode
                                               t ................................ Command Destination Thread Id
                                               r ................................ Reserved
[N words of command data...] .......... Optional EDS Command Words 1 through N
                                               Containing EDS Command specific data, where N must
                                               be less than or equal to 27 words (54 bytes).
```

The *packet length* field contains the length of the entire packet, in bytes, minus the size of the primary CCSDS Header (6 bytes), minus 1. Note that, since command packets can be no longer than 64 bytes, the top 10 bits of this field are zero.

The **FDS-EDS Interface Control Document** states that the ground software may not be capable of ensuring that the *sequence number* increments properly with each packet sent to the EDS, and recommends that the software ignore this field. The EDS Flight Software echos the entire command packet, and, therefore, this field, back to the ground when acknowledging commands, as part of the acknowledgement data. Although the value may not increment in sequence, it still may provide a method of uniquely identifying the command packet in a series of up to 16,385 command packets.

3.2.2 Telemetry Packets

All telemetry packets, sent by the Flight Software to the FDS, start with a CCSDS compliant header, followed by a secondary header, followed by EDS specific information. The Flight Software ensures that all packets sent to the FDS are sent using 1024 bytes. Each 16-bit word of the telemetry packet is retrieved by the DMA in little-endian format (least significant byte first). The FDS truncates the received packets, using the length specified in the CCSDS header, before transmitting them to the ground.

[v,v,v,t,h,a,a,a,a,a,a,a,a,a,a,a] .... CCSDS Header Word 0
v .... Version = 000
t .... Type = Telemetry = 0
h .... Secondary header present = 1
a .... Application Id
[g,g,s,s,s,s,s,s,s,s,s,s,s,s,s,s] .... CCSDS Header Word 1
g .... Segment Flag = None = 11
s .... Sequence Number
[l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l] .... CCSDS Header Word 2
l .... Packet Length
[t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t] .... Secondary Header Word 0
t .... Bits 16-31 of time in seconds
[t,t,t,t,t,t,t,t,t,t,t,t,t,t,t,t] .... Secondary Header Word 1
t .... Bits 0-15 of time in seconds
[e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e] .... Secondary Header Word 2
e .... Bits 16-31 of EA Run Id
[e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e] .... Secondary Header Word 3
e .... Bits 0-15 of EA Run Id
[r,r,r,r,r,r,r,r,d,d,d,d,d,d,d,d] .... EDS Telemetry Word 0
r .... Reserved
d .... Data Stream Id
[N words of telemetry data...] .... Optional EDS Telemetry Words 1 through N Containing EDS Data Stream, where N is less than or equal to 504.

The FDS supports an aggregate rate, from all 8 EA's, of about 500Kbits per second. The time between the start of two consecutive packets sent by a single EA, given that all EA's are sending packets at the same time, is approximately 131ms (i.e. about 62Kbits/sec. from a given EA). Given a situation where only one EA is sending packets, the time between two consecutive packets is approximately 16ms (i.e. about 512Kbits/sec. from the sending EA). Please refer to Section 4.1.10 for a description of how this applies to the science telemetetry rate on a given EA.

NOTE: All EDS telemetry data which contain 80286 memory addresses will send the address using the 80286 native segment:offset format. This format consists of two 16-bit words. The first word will contain the "offset" portion of the address, and the second word will contain the "segment" portion.

3.3 Event Analyzer Back End RAM Memory Map

This section provides a brief description of the RAM Memory layout within a given Event Analyzer (EA). The hardware supplies the Event Analyzer's Back End with 256K of static RAM. Upon boot, each EA copies the contents of the System Manager board's ROM and Protected RAM into the lower 192K of this space. The top 64K is used as an intermediate science buffer when the EA copies data from the Front End during an observation. The following illustrates this layout:

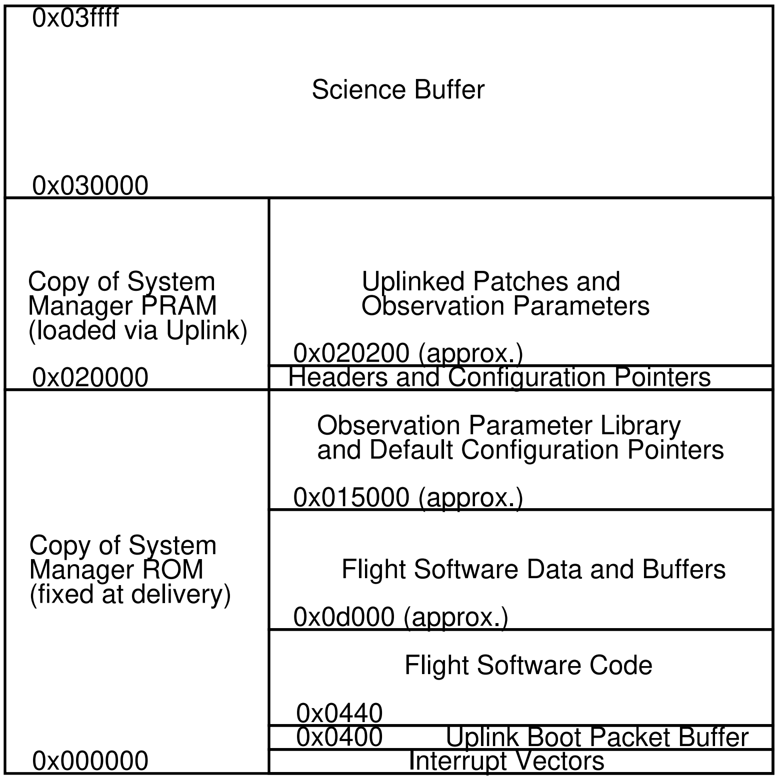
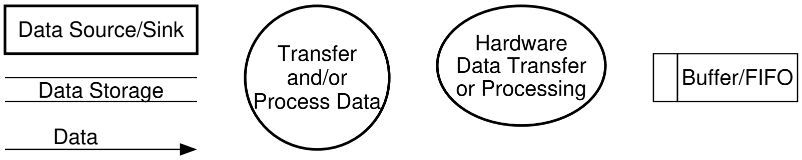


Figure 6 EA RAM Memory Map

4 Design Description

4.1 Data Description

This section uses the following picture elements to describe the data flow of the EDS:



4.1.1 Event Analyzer Data Context

The following diagram illustrates the overall data flow of the EDS:

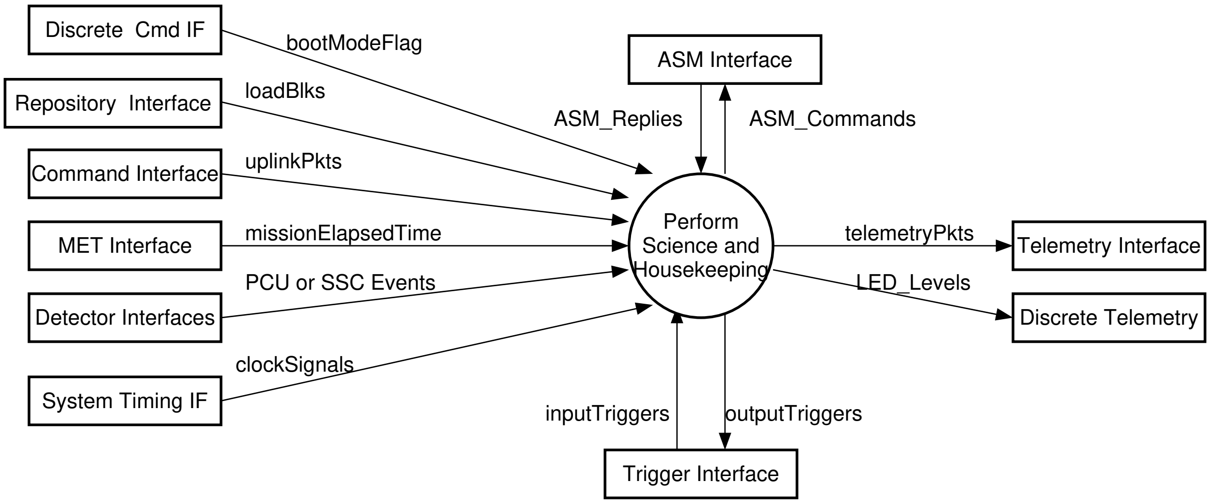


Figure 7 Event Analyzer Context Diagram

4.1.2 Level 1 Data Flow

The following illustrates the overall data flow within the Flight Software. The bubble numbers identify a particular data processing task and are referred to later in this document.

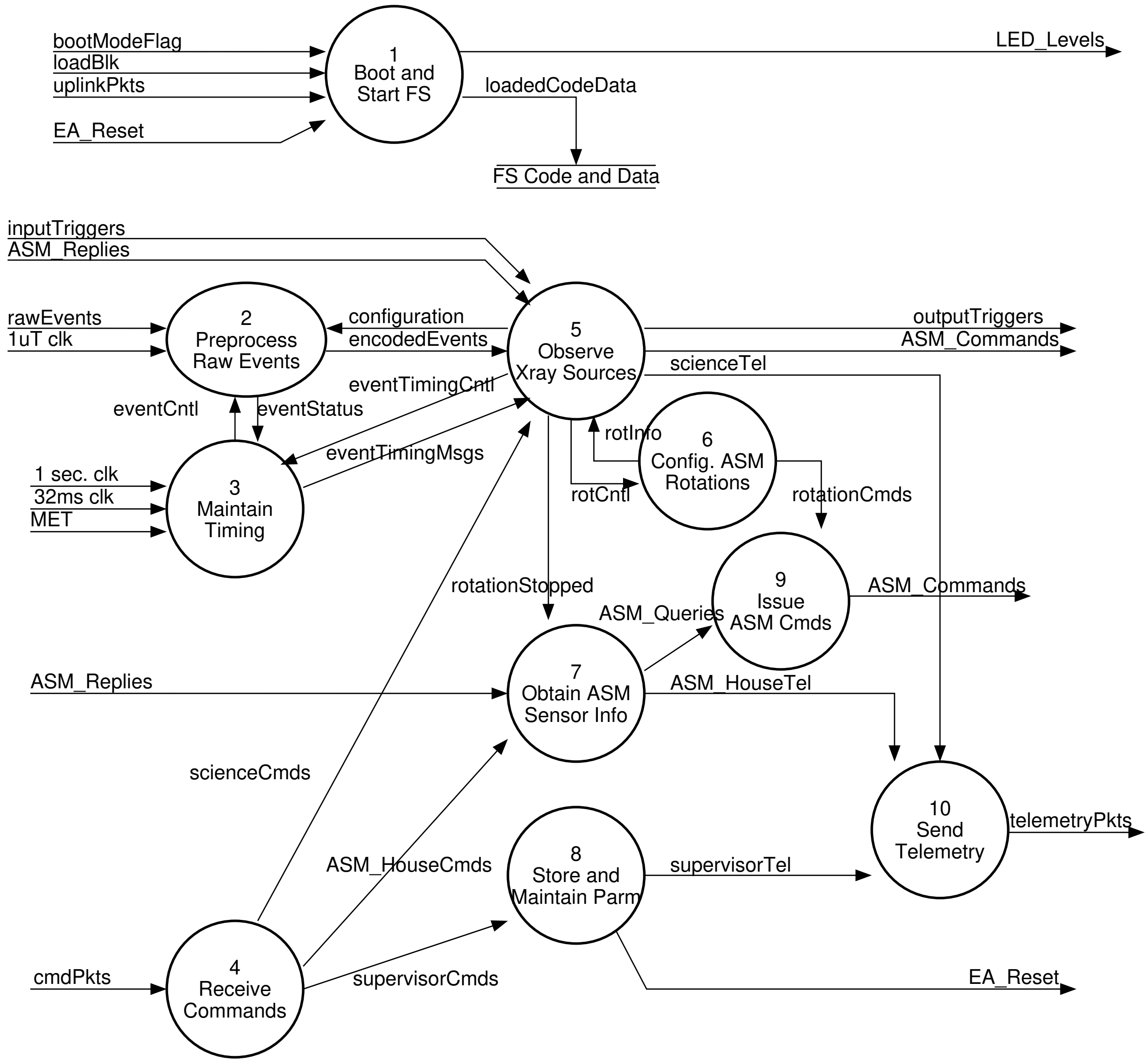


Figure 8 EDS Flight Software Level 1 Data Flow

4.1.3 Threads

The Flight Software consists of 3 threads, represented by the Level 1 process bubbles 5, 7 and 8. Each thread is responsible for executing and replying to commands addressed to it by the uplinked command packets.

4.1.4 Telemetry Data Flow

The following diagram illustrates the overall flow of telemetry within the Flight Software. Bolded process bubbles represent the actions taken by the Level 1 process bubbles 5, 7 and 8, with respect to generating telemetry:

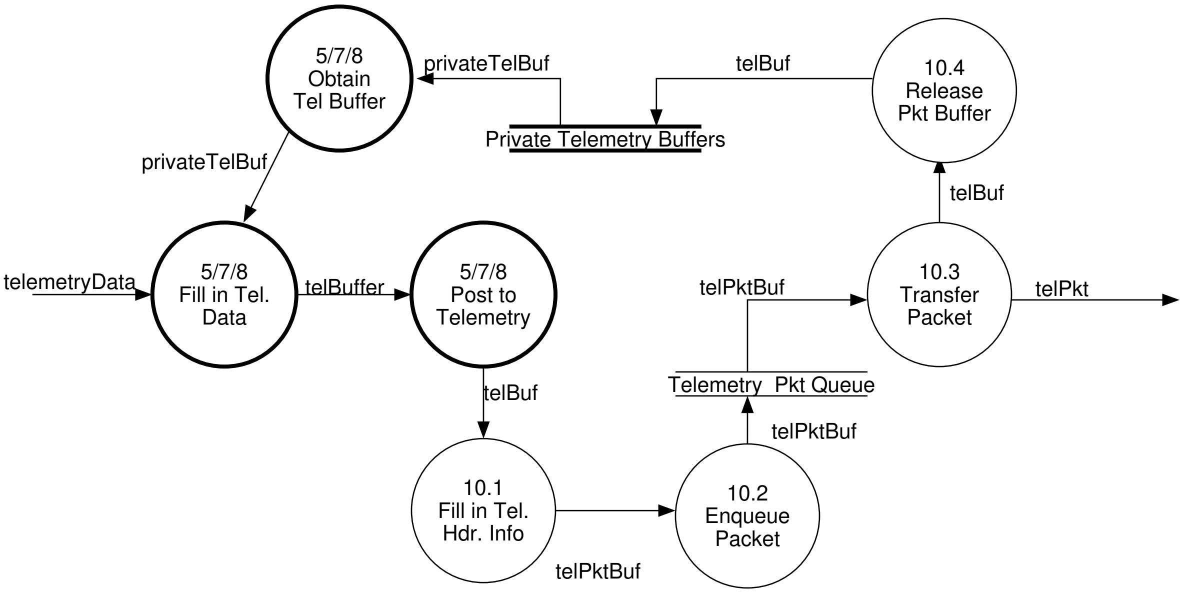


Figure 9 Telemetry Data Flow

Each thread responsible for generating telemetry, initializes and maintains one or more pools of telemetry buffers. When it needs to send telemetry, it obtains a free buffer from one of its private pools, fills in the data portion of the telemetry buffer, and posts the buffer to the telemetry subsystem. The telemetry subsystem then fills in the packet's header information and enqueue's the buffer onto a queue of packets waiting to be telemetered. If the queue is empty, the telemetry subsystem starts transferring the packet, using the DMA's downlink channel. Once the packet has been transferred, the hardware causes a Downlink interrupt, indicating that the packet has been transferred. If there are more packets in the queue, the telemetry subsystem then starts the transfer of the next packet. Once a packet has been transferred, the telemetry subsystem releases the packet's buffer to the pool from whence it came. If the pool is empty when the packet is released, the pool's owner is notified that a packet is available.

4.1.5 Command and Acknowledgement Data Flow

The following diagram illustrates the data flow of command and command acknowledgment packets. Bolded process bubbles represent the actions taken by the Level 1 process bubbles 5, 7 and 8, with respect to commands and acknowledgments.

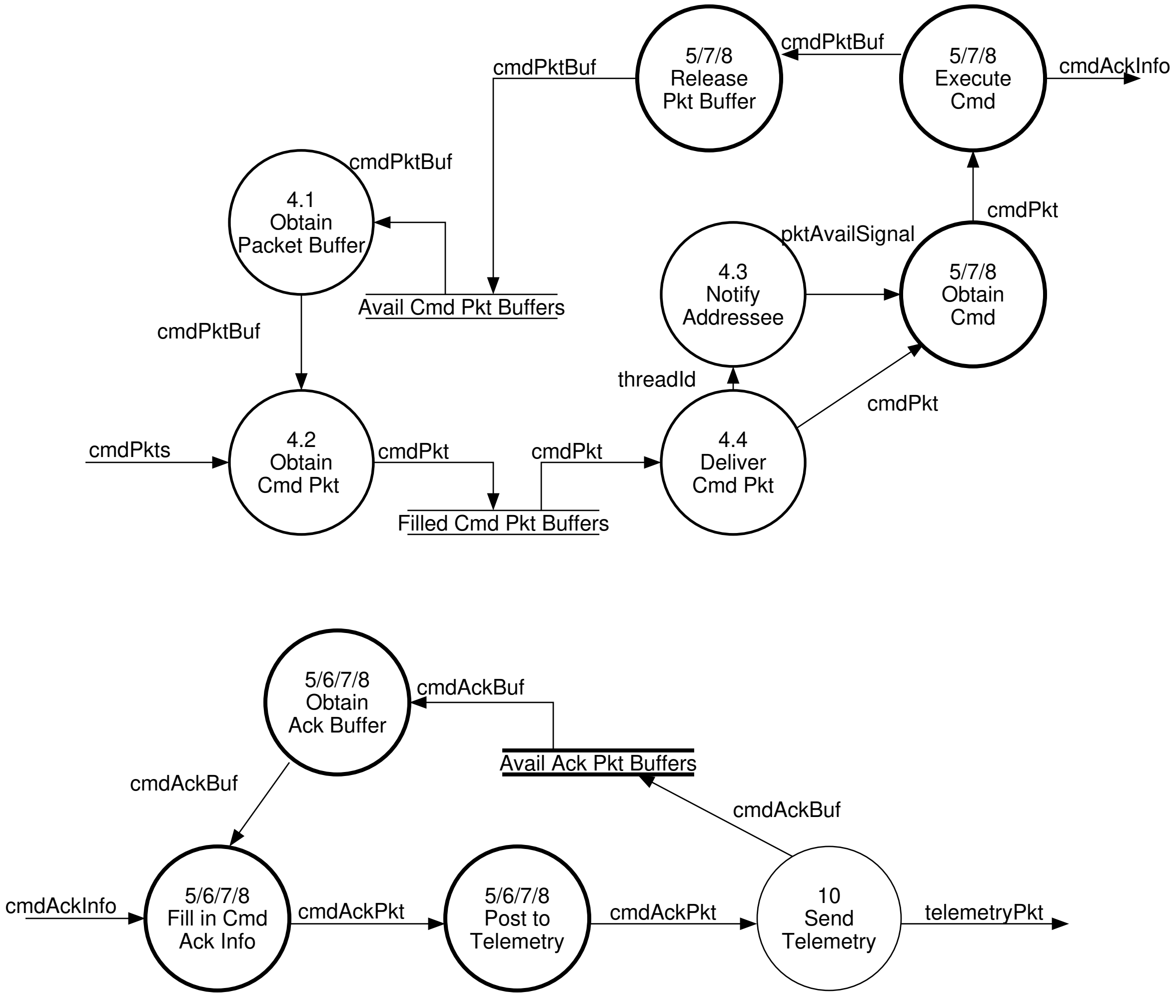


Figure 10 Command and Acknowledgement Data Flow

Commands are received using the DMA's uplink channel. The uplink manager initializes a pool of uplink packet buffers. It then sets up the DMA controller to transfer an uplink packet and waits for a command packet to be received. When the packet arrives, the hardware interrupts the system, indicating that an uplink transfer is complete. The uplink manager enqueue's the received packet buffer onto its queue of unprocessed commands. The uplink manager then notifies the thread addressed by the packet that a command is available. The thread then retrieves the command, and the uplink manager notifies the thread identified by the next uplink packet in the queue. Once the thread has finished processing the command, it releases the uplink buffer back to the buffer's pool.

Sometime during the thread's execution of a command, an acknowledgment must be generated. The thread obtains an acknowledgment buffer from a pool of buffers, fills in the response to the command, and posts the packet to be telemetered by the telemetry subsystem. Once the packet has been transferred, the telemetry subsystem releases the acknowledgment buffer to the buffer's pool.

As the uplink manager receives and enqueues each command packet, it obtains a new uplink buffer and configures the DMA for another transfer. If the system runs out of uplink buffers, due to slow command execution, it sets up the DMA to transfer packets into a dummy buffer, and records the number of discarded commands in the software housekeeping database (not shown in Figure 10).

4.1.6 Parameter Block Data Flow

The following diagram illustrates the overall flow of parameter blocks, where the bolded process bubbles represent the actions taken by the Level 1 process bubble 5, with respect to parameter blocks.

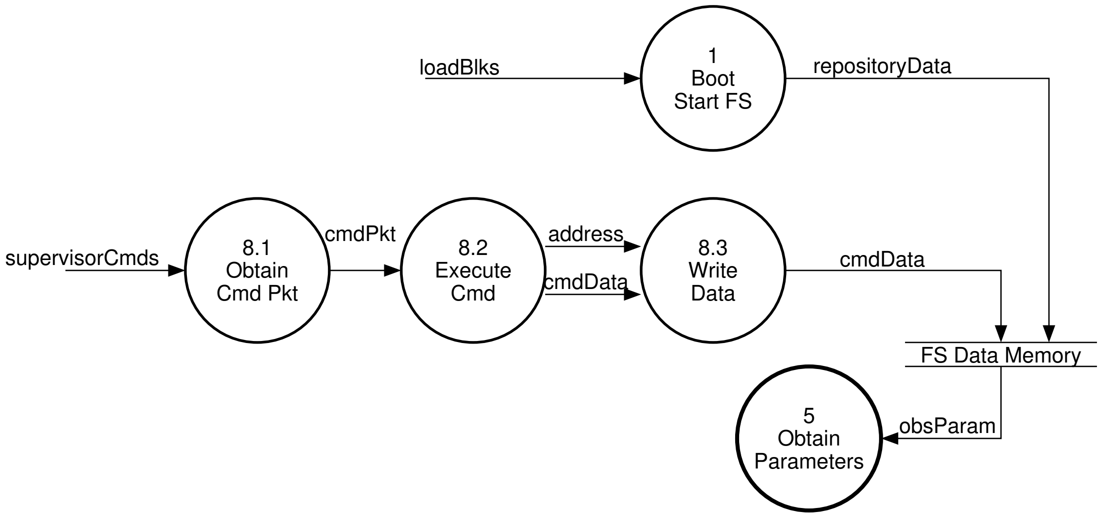


Figure 11 Parameter Block Data Flow

The ground loads parameters into the Flight Software by directly writing the data blocks into the EA's memory. This is accomplished three ways. First, all standard parameters, known before delivery, are stored in the ROM Repository. When the EA boots, these parameters are copied into the EA's RAM. Second, all non-standard or modified standard parameters can be written into the EA sub-group System Manager PRAM Repository. When the EA boots, these parameters are also copied into the EA's RAM. Third, the Flight Software is commandable to write information directly into the EA's RAM. When the Science Thread starts, it retrieves a pointer to the observation's parameters from the loaded image. The thread then directly retrieves the parameters from RAM.

4.1.7 Software Housekeeping Data Flow

The Flight Software provides facilities for any module to report various operating statistics. Each statistic is represented using a 16-bit integer value, and is associated with an Id code. The statistics are reported using a global utility function. The FS (process bubble 8) periodically (i.e. once every several MajorTicks, where a MajorTick = 1 second) obtains the accumulated statistics and posts them to the telemetry subsystem. The following diagram illustrates the overall statistic data flow:

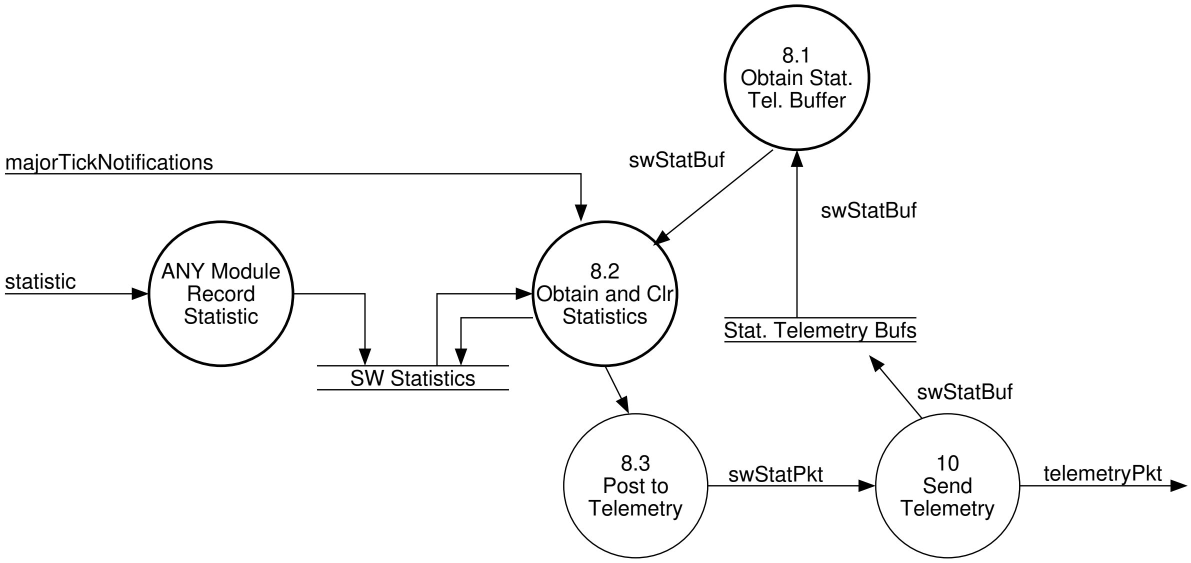


Figure 12 Software Housekeeping Data Flow

4.1.8 ASM Housekeeping Data Flow

The following diagram illustrates the ASM engineering sensor data flow:

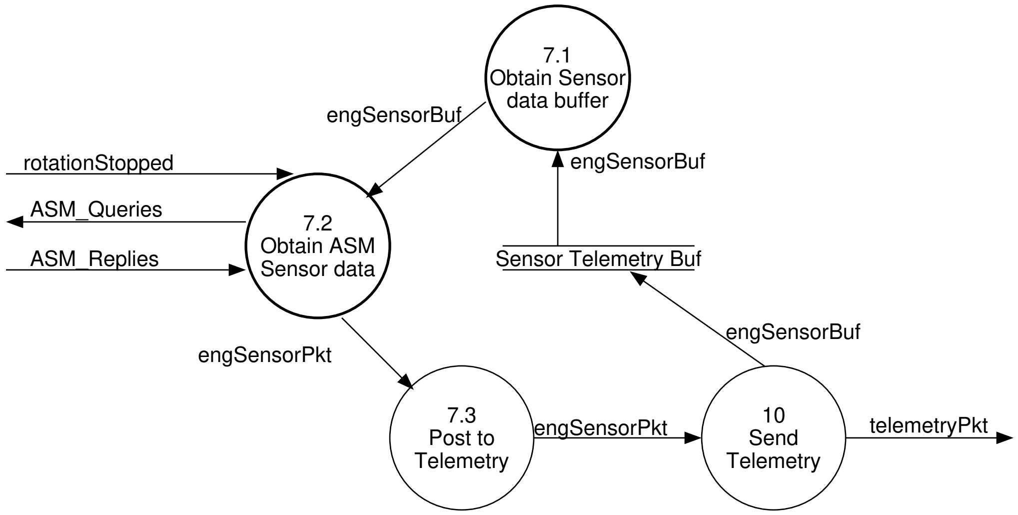


Figure 13 ASM Housekeeping Data Flow

The Flight Software periodically obtains and telemeters engineering sensor values from the ASM, using an ASM Housekeeping manager. The manager obtains an Engineering Sensor telemetry packet from a private pool of sensor buffers, and waits for indication that the ASM interface is available. The interface is considered available when no camera rotations are being performed. Once the interface is available, the manager sequentially queries the ASM systems, obtains the engineering sensor values, and stores the results in the telemetry buffer. Once all of the sensors have been read, it posts the buffer to the telemetry subsystem. Once the buffer has been transferred, the telemetry subsystem returns the buffer to the pool.

4.1.9 Science Data Flow

The following diagram illustrates the overall flow of the Science Data:

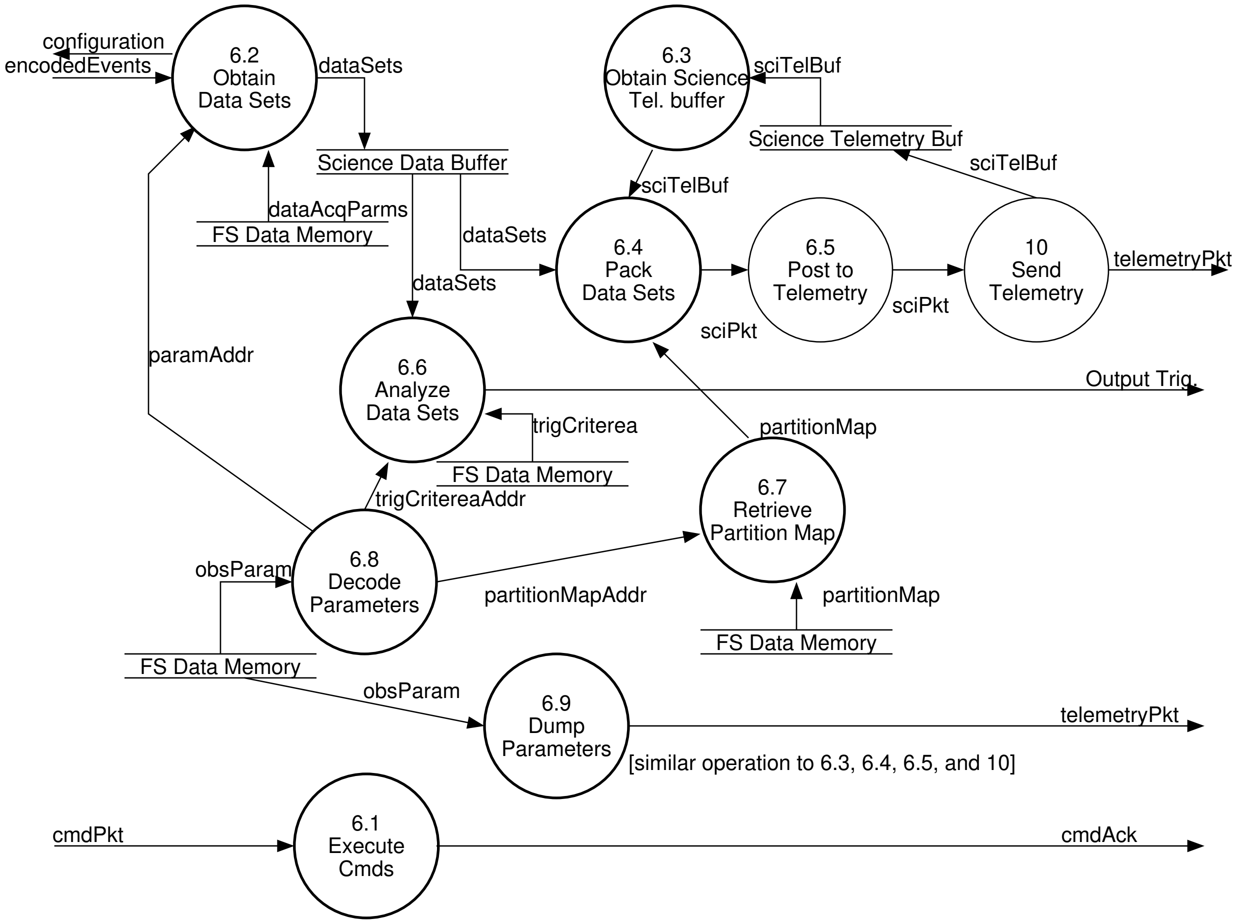


Figure 14 Science Data Flow

Science processing consists of five main types of operations; setup, dump, data acquistion, data analysis, and telemetry generation. During startup, the science processing retrieves a pointer to the observation’s configuration parameters from RAM, decodes the referenced parameters, and initializes the event processing hardware. It then dumps the parameters to the ground. Science processing then uses the parameters to control the acquisition and processing of the events. The processed events form event data sets, or "partitions". These partitions are obtained from the DSP and stored in a partition data area in the CPU memory. This data is then compressed, via a partition map, and packed into telemetry packets, which are then posted to the telemetry subsystem. If the science processing is configured to analyze the data, it does so as the telemetry packets are transferred out of the EDS. Refer to Section 4.1.10 for an overall description of event processing.

4.1.10 Event Processing Hardware Data Flow

This section describes the data flow of events as they are processed by the Event Processing Hardware:

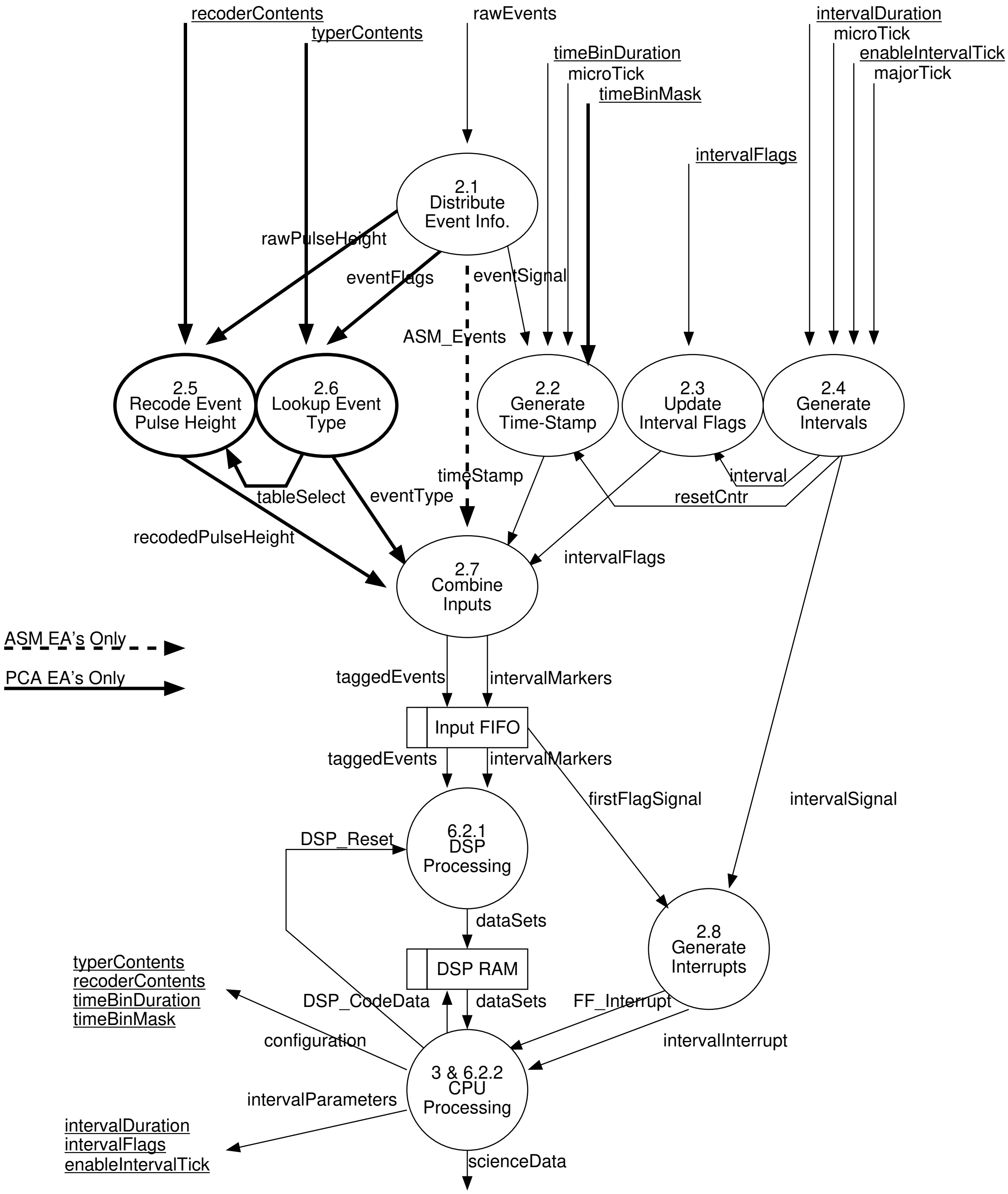


Figure 15 Event Processing Hardware Data Flow

The event processing hardware comes in two flavors, one for the PCA, and the other for the ASM. On the PCA, the event processing hardware consists of the following: event type lookup table (Typer), event pulse height lookup table (Recoder), event timing circuitry and a DSP. On the ASM, the event processing hardware consists of only the event timing circuitry and a DSP. The Flight Software is responsible for configuring and controlling the event processing hardware during a given observation.

Event processing consists of four main steps:

1) Configure the hardware .... This involves resetting the DSP, configuring the Typer and Recoder Lookup tables, and setting the time-bin duration and mask.
2) Dump the parameters .... This involves telemetering all of the parameters that are used by the observation.
3) Run the observation .... This involves scheduling the intervals, obtaining the data sets from the DSP, analyzing the data sets and triggering another EA or HEXTE if required, and packing and telemetering the obtained data.
4) Check the configuration .... After an observation has ended, the event processing software tests the integrity of the tables and parameters used to process the events, and telemeters this information to the ground.

The Flight Software supports a variety of data acquisition modes (Science Modes). Each of the modes is specified by the **EDS-PCA Science Requirements** or by the **EDS-ASM Science Requirements**. These modes shall include:

Binned Mode .... Build histograms of event times, types and energies. Support data set analysis and burst triggering.
Pulsar Fold Mode.............................. Build histograms of event phase, types and energies.
Event Encoded Mode........................ Accumulate event words sequentially.
Single Bit Mode .... Generate a bit map of time-bins and event occurences.
Burst Catcher Mode .... Obtain high-time resolution data when triggered by another EA or by HEXTE.
FFT Mode .... Generate Power Density Spectra and Cross-Spectra.
Autocorrelation Mode....................... Generate autocorrelation data sets (Level 2 requirement, not currently documented).
Position Histogram Mode .... Build histograms of measured event positions (ASM)
Multiple Time Series Mode .... Build several histograms of event properties (ASM)
Event by Event Mode .... Accumulate event words sequentially (ASM)

Science processing occurs in a pipe-lined fashion. Hardware control is performed on an interval by interval basis. At each interval boundary, the hardware updates from a set of flags, set by the Flight Software during the previous interval. Initially, the Flight Software sets up the event processing hardware and schedules the first interval of the observation. This interval is started by the hardware within 1 microtick of the next MajorTick pulse. The software schedules the first interval such that the starting MajorTick pulse coincides with the next MET value evenly divisible by 16. When the interval starts, the hardware writes an interval marker word into the DSP's Input FIFO, and generates an interval interrupt on the Back End of the EA. The Back End interval interrupt handler then sets up the parameters for the next interval. The time taken to handle the interrupt and schedule the next interval must be less than the duration of the current interval. Since the **EDS-PCA Science Requirements** imply that interval durations may vary as often as once every 5ms (Pulsar Fold Mode), the Flight Software must not take more than 1ms per interval interrupt to ensure that no more than 20% of the CPU time is taken up by this single interrupt handler.

The CPU controlled flags which are updated at every interval boundary are as follows:

Sequence Active .... When asserted, the event processing hardware starts writing processed events to the DSP's Input FIFO upon the start of the interval. When deasserted, the event processing hardware stops writing processed events at the start of the next interval.

First-Flag .... When asserted, a First-Flag interrupt is generated by the hardware when the DSP reads, from its Input FIFO, the interval mark associated with the start of the next interval. When deasserted, no First-Flag interrupt is generated when the associated interval mark is read. The interrupt is used by the Flight Software to ensure that the DSP has processed all of the events from the previous interval before it copies the Science Data from the DSP memory to the CPU memory.

Interval Constant............................... This value is "or'd" into the processed event words which were time-stamped during the associated interval, and is used by some Science Modes to select regions of the DSP's memory.

In addition to these CPU controlled flags, the event processing hardware automatically updates the following flag at each interval boundary:

Spillage .... If the DSP's Input FIFO is full at any point during a given interval, the hardware sets this flag at the start of the next interval. This flag is readable by the CPU.

Each of the science modes are similar in their overall operation. The following shows a simplified science processing data flow, for the purpose of discussing the timing and throughput of the science data:

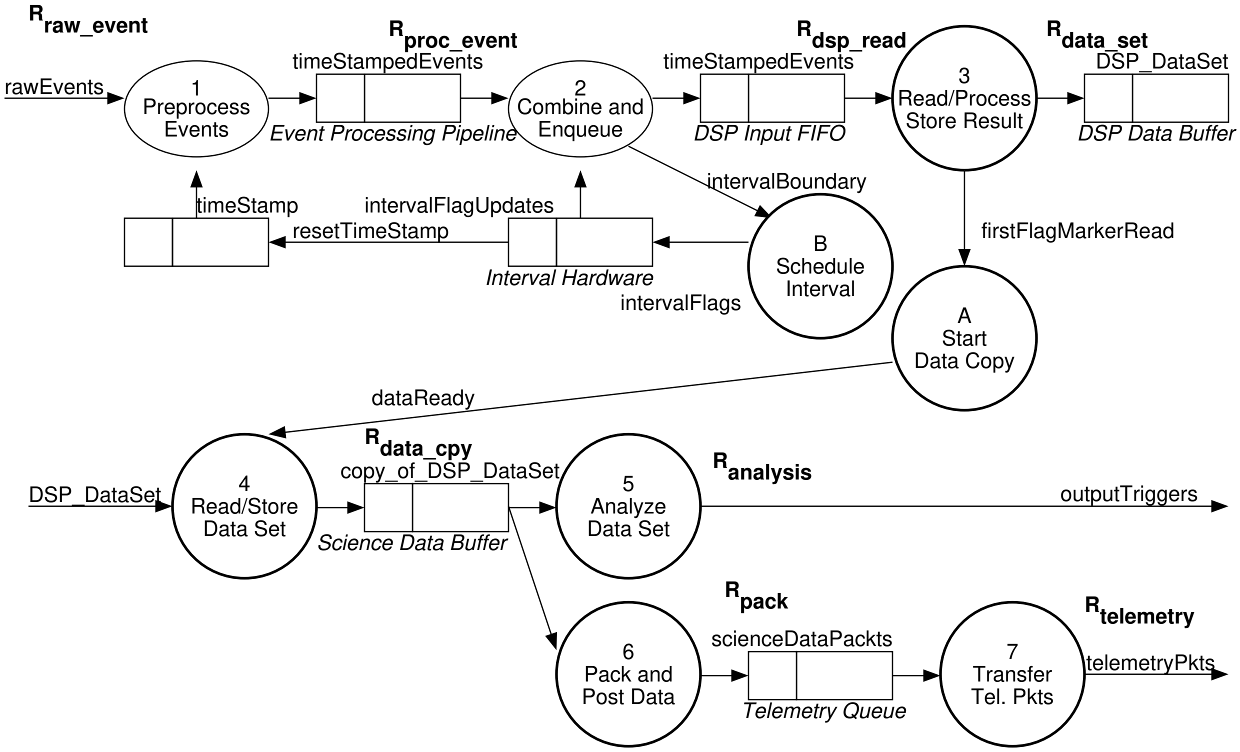


Figure 16 Simplified Science Processing Data Flow Diagram

Figure 16 indicates processing rates at various stages within the science processing data flow. The rates and their meanings are as follows:

R_{raw_event} .... This represents the raw event rate coming from the PCU's or SSC's. Each PCU can produce up to 143K events/second, times 5 PCU's gives a maximum raw event rate of 714K events/sec. Normally, the raw event rate is determined by which X-ray source the system is observing. For most sources the average event rate is known when configuring the EDS.

R_{proc_event} .... This represents the processed event rate. This rate is governed by the raw event rate, and the number of events which are accepted by the event processing Typer and Recoder lookup tables (configurable using the Flight Software).

R_{dsp_read} .... This represents the rate at which the DSP reads and processes events words from its Input FIFO. This rate is extremely mode dependent.

R_{data_set} .... This represents the rate at which the DSP produces science data sets (partitions). This value depends on the particular mode. For most modes, it is proportional to the interval duration used for the observation. The data rate for a given data set depends on this value and the number of bytes in the data set.

R_{data_cpy} .... This represents the rate at which the CPU can copy DSP data sets into its Science Data Buffer, and, if required by the mode, reset portions of the DSP's data area. This rate is dependent on the code required to perform the copy/reset, and a variable amount of system overhead (interrupt processing, etc.). Typically, this rate will remain fairly constant.

$R_{analysis}$ .... This represents the rate at which the CPU can analyze the science data sets. This rate is dependent on the criteria and code used to analyze the data sets, and a variable amount of system overhead.

R_{pack} .... This represents the rate at which the CPU can pack the science data into telemetry packets. This rate is dependent on the code required to pack the data, and a variable amount of system overhead (interrupt processing, etc.). Typically, this rate will remain fairly constant.

$R_{telemetry}$ .... This represents the rate at which the CPU can transfer telemetry packets to the FDS. This rate can be governed by a bit in the parameter block of the running Science Mode to be 1 packet/sec., or is governed by the FDS (and hence depends on the rates out of the other EAs).

The following lists certain rate mismatch conditions, their effects, and how they should be handled:

$R_{raw\_event} > R_{proc\_event}$ .... The hardware guarantees that it can process events faster than the PCA or ASM SSC's can deliver them, so this condition will never occur (see **XTE-EDS Hardware Functional Description and Requirements**, Sections 2.1.4.12, 2.1.4.14).

$R_{proc\_event} > R_{dsp\_read}$ .... If this condition is sustained for too long, the DSP's Input FIFO will overflow. This condition is indicated by the hardware via a Spillage Flag. When this condition arises, the hardware will stop writing events into the Input FIFO until the start of the next interval. If this condition occurs close enough to the start of the next interval, such that the Input FIFO remains full at the start of the next interval, spillage will be indicated for that interval as well, and no events will be written for that interval. If the DSP reads at least one event during this second interval, the condition won't not span more than 2 successive intervals. If, however, the DSP doesn't read an event, the condition will persist.

$R_{dsp\_read} > (R_{data\_set} * \text{Data Set Size})$ .... This condition is very mode dependent. In some modes, the DSP will detect and indicate this condition to the CPU via some CPU readable flag or "dropped event" count. In other modes, the DSP has no way of detecting the occurrence, and the effect is only detectable by analyzing the produced data set. In either case, the DSP will always continue to read the Input FIFO and possibly discard events to prevent an Input FIFO overflow condition.

$(R_{data\_set} * \text{Data Set Size}) > R_{data\_cpy}$ .... This condition indicates that the DSP is producing data sets at a rate faster than the CPU can copy them and, if required, reset the DSP's data area. Since the data set size and rate at which the data sets are produced is configurable by the ground, this condition indicates that the mode is improperly configured, or that some ground initiated action is causing too much system overhead. In either case, the CPU will indicate to the ground that there is a problem. Whether or not the CPU can sample data sets in this case is mode dependent. In some configurations, this condition causes the EA to issue a panic message and reset.

$R_{data\_cpy} > R_{analysis}$ .... This condition indicates that data sets are being produced faster than the CPU can analyze them. This indicates that the ground has configured the system to produce data sets faster than can be analyzed, given the mode of analysis (burst trigger criterea), or that some ground initiated action is causing too much system overhead. In this case, the CPU will discard complete data sets from the analysis until the current data set has been analyzed, tracking the number of data sets which were discarded. The CPU will also indicate the problem to the ground. If the CPU is required to reset portions of the DSP's data area, it will continue to do so.

$R_{data\_cpy} > R_{pack}$ .... This condition indicates that data sets are being produced faster than the CPU can pack them into telemetry packets. This indicates that the ground has configured the system to produce data sets faster than can be packed into telemetry packets, or that some ground initiated action is causing too much system overhead. In this case, the CPU will discard data sets, tracking the number of data sets which were discarded, and if required, continue to reset portions of the DSP's data area. This allows the system to obtain complete data sets, but with gaps between data sets. The CPU will indicate this condition to the ground.

$R_{pack} > R_{telemetry}$ .... This condition indicates that telemetry packets are being filled faster than they can be transferred out of the EDS. If this condition continues for too long, the EDS will temporarily run out of telemetry buffers. In this case, the CPU will continue to telemeter the current data set, as telemetry buffers become available. It will discard new data sets, tracking the number of data sets which were discarded, and if required, continue to reset portions of the DSP's data area. This allows the system to obtain complete data sets, but with gaps between data sets. The CPU will indicate this condition to the ground.

Historically, the telemetry rate ($R_{telemetry}$) has been assumed to be much less than the rate at which the CPU can copy the DSP's data (R_{data_cpy}), the rate at which the CPU can analyze the data ($R_{analysis}$), and the rate at which the CPU can pack data into telemetry packets (R_{pack}). Using these assumptions, the data set rate (R_{data_set}) was bounded by the telemetry rate. These assumptions don't apply when the system is configured to analyze the data without telemetering it. In this case, one can assume that the data analysis rate is the bounding factor.

With the new spacecraft, the telemetry rate ($R_{telemetry}$) is very fast (on the order of 512Kbps, or 64 full telemetry packets/second). The utilized telemetry tends to be bounded by the rate at which data is produced (R_{data_set}). Since the data set rate is configured by the ground, by setting interval durations and data set sizes, the ground should use these parameters to achieve the desired telemetry rate.

The CPU performs all data copy, analysis and packing operations. These operations, in addition to interval scheduling and system overhead, all affect the CPU utilization, and are interrelated.

Given that it is possible that all 8 EA's may try to send their partitions at the same time, and that one or more of the EA's are trying to send partitions at a high rate, the system is configurable by the ground to govern the slower EA's and limit how many telemetry packets they send within a certain period of time. This ensures that the high rate EA's are serviced by the FDS in time to maintain their desired data rate (see Section 3.2.2).

4.2 Flight Software Module Structure

The Flight Software consists of a set of software modules. These modules can be grouped into three types; Thread, Library, and Kernel modules. Thread modules are responsible for command execution and the bulk of the data acquisition. Library modules are responsible for providing high-level services to the threads, such as table decompression. Kernel modules are responsible for providing interfaces to the Flight Hardware, and for providing some software utility functions. In the following descriptions, if a particular module is responsible for a process shown in Figure 8, the process bubble number is listed with the description of the module.

4.2.1 Thread Module Descriptions

The following lists all of the Thread Modules contained in the Flight Software:

Supervisor Thread (Data Flow 8)	The Supervisor Thread is responsible for maintaining the health of the Flight Software and for periodically providing the ground with statistics relating to the operation of the system. This thread is commandable to read, write and execute sections of memory. This thread is also responsible for periodically resetting the Watchdog timer.
Housekeeper Thread (Data Flow 7)	The ASM Housekeeper Thread is responsible for obtaining and telemetering the ASM's Engineering Sensor values.
Science Thread (Data Flow 5)	The Science Thread is responsible for acquiring, processing, and telemetering xray event data. It is commandable to stop event processing, and suspend and resume telemetry generation. The event processing mode and parameters are ground configurable via a configuration table contained in the PRAM Repository.

4.2.2 Library Module Descriptions

The following lists all of the Library Modules contained in the Flight Software:

Typer Builder Library	The Typer Builder Library Module is responsible for generating the contents of the Typer Lookup Table from sets of ground loaded parameters, for supplying the length of the parameters, and for checking the integrity of the Front End's Typer Lookup Table after an observation has ended. It has knowledge of the parameter formats and of the Typer Lookup Table inputs and outputs.
Recoder Builder Library	The Recoder Builder Library Module is responsible for generating the contents of the Pulse Height Recoder Lookup Table from sets of ground loaded parameters, for supplying the length of the parameters, and for checking the integrity of the Front End's Recoder Lookup Table after an observation has ended. It has knowledge of the parameter formats and of the Recoder Lookup Table inputs and outputs.
Partition Map Library	The Partition Map Library Module is responsible for managing the ground loaded Partition Map, for supplying the length of the Partition Map, for packing data according to the loaded map, and for checking the intergity of the map after an observation has ended. The Partition Map Module is used by the Science Thread to pack science data before it is telemetered to the ground.

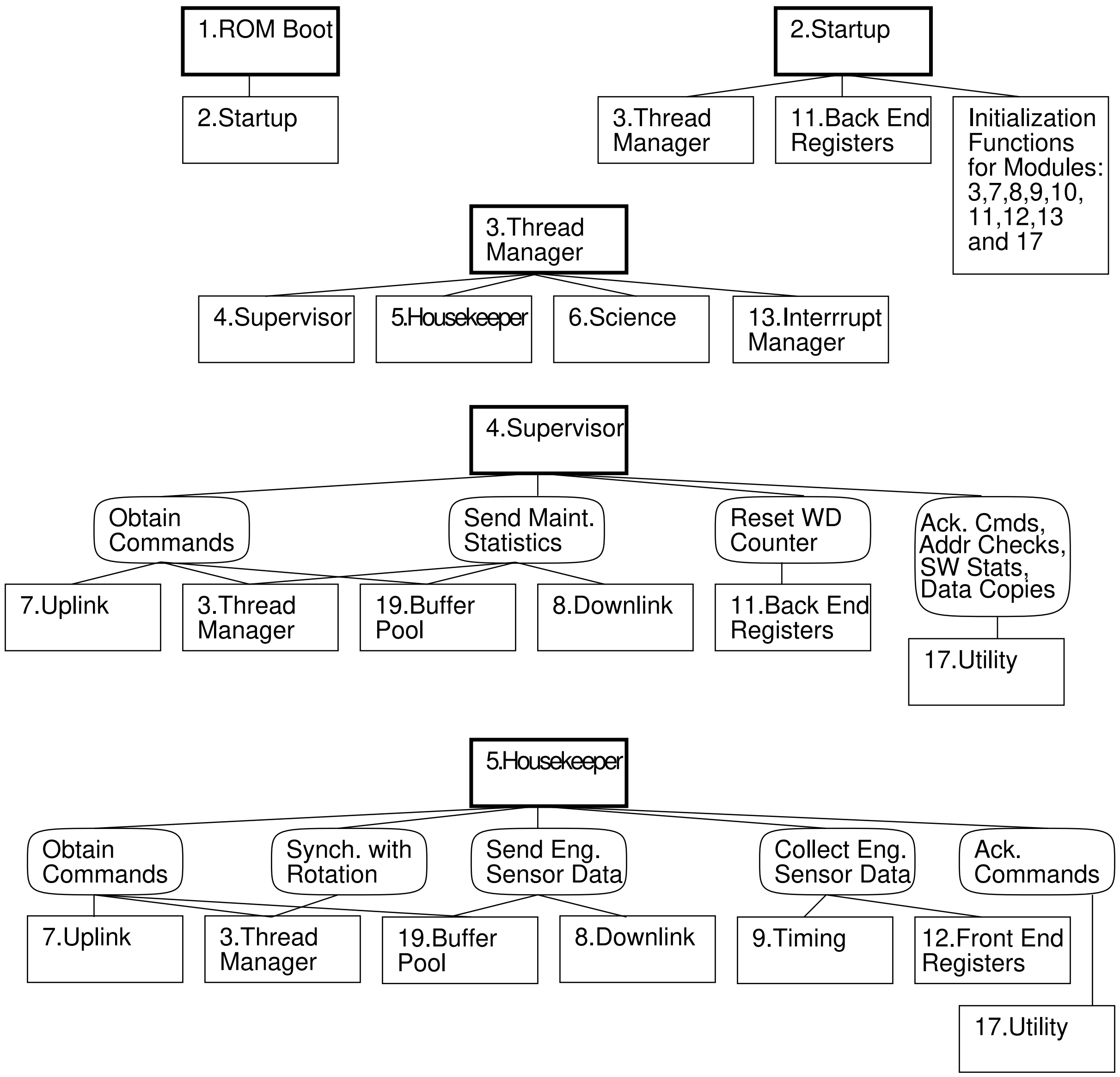
4.2.3 Kernel Module Descriptions

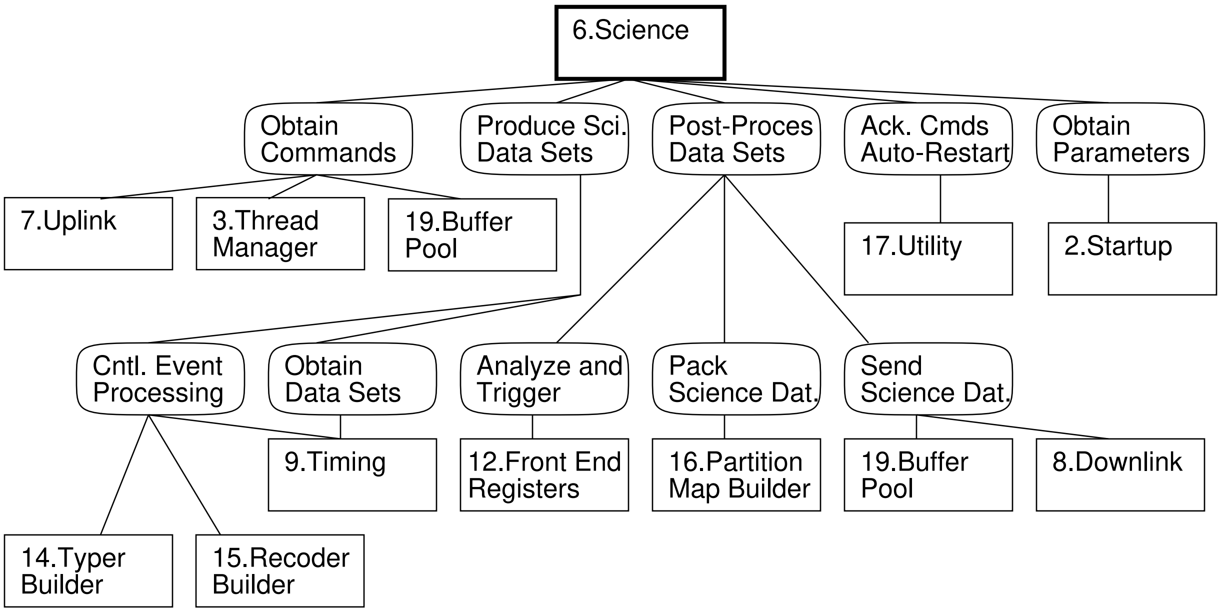
The following lists all of the Kernel Modules in the Flight Software:

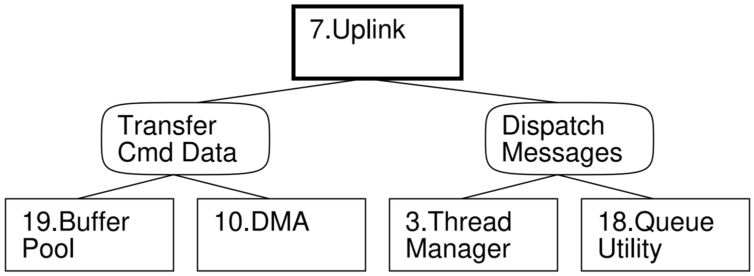
ROM Boot Module (Data Flow 1)	The ROM Boot Module resides in each EA's ROM. It is responsible for loading code and data from either the ROM/PRAM Repository, or from the Uplink DMA channel. Once the code is loaded, it is responsible for initiating the execution of the code. This module provides rudimentary diagnostic information during the boot using the discrete LED signals to indicate progress to the ground.
Startup Module (Data Flow 1)	The Flight Software Startup Module is responsible for installing loaded patches, initializing all of the FS modules, and starting the multitasking (Thread Manager) executive. Once the FS is running, the Startup Module provides the system's modules with configuration information stored in the ROM or PRAM Repository.
Thread Manager	The Thread Manager is responsible for controlling thread execution, using "event" notifications to schedule which threads to run.
Uplink Module (Data Flow 4)	The Uplink Module is responsible for obtaining command packets, using the DMA Module, and for dispatching the received command packets to their addressed thread.
Downlink Module (Data Flow 10)	The Downlink Module is responsible for obtaining telemetry data from the threads, filling in telemetry header information, and for using the DMA Module to transfer telemetry packets to the FDS.
Timing Module (Data Flow 3)	The Timing Module is responsible for managing all timing related activities in the Flight Software. It maintains the current reference time (MET), manages the event processing timing hardware, and provides real-time interrupt call-back facilities.
DMA Module	The DMA Module is responsible for managing the Uplink and Downlink DMA channels on the DMA controller.
Interrupt Manager	The Interrupt Manager Module is responsible for managing the interrupt controller. It provides interrupt call-back facilities for all interrupts in the system. It also provides the ability to disable and re-enable interrupts in a nested fashion.
Back End Registers	The Back End Registers Module provides functions which read and write all of the Back End related hardware registers (except DMA/Interrupt)
Front End Registers	The Front End Registers Module provides functions which read and write all of the Front End (observation) related hardware registers (except DSP RAM and Lookup Table RAM).
Buffer Pool	The Buffer Pool Module provides utility functions which manipulate sets of equally sized buffers. It also provides facilities to enqueue and dequeue buffers to and from client defined queues, using the Queue Utility functions.
Queue Utility	The Queue Utility Module provides functions to manipulate element queues.
Utility Module	The Utility Module provides the Flight Software with machine specific utility functions.
Rotation Utility	The Rotation Utility Module is responsible for issuing rotation commands to the ASM, and for providing the Flight Software with the duration of each rotation.

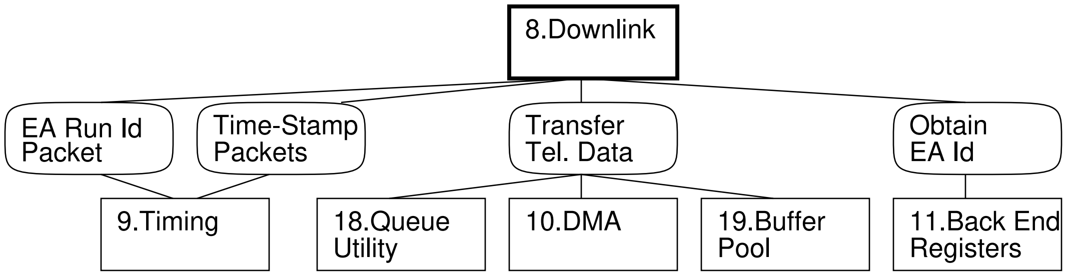
4.2.4 Flight Software Structure

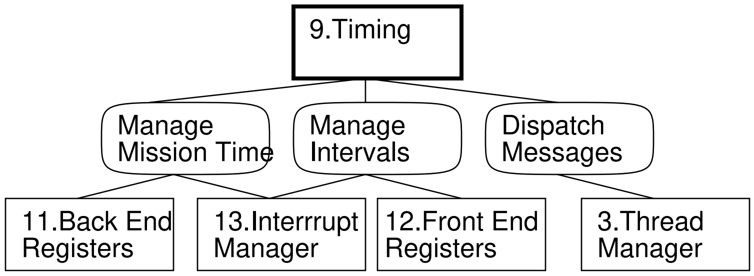
The following diagram attempts to illustrate the overall structure of the Flight Software. Each rectangular bolded box represents the module being described, oval boxes represent some of the internal operations of the module, and the rectangular unbolded boxes represent the modules which are called or controlled by the module being described. Section 4.3.x (where "x" represents the module number in each box below) describes, in more detail, each of the module's services, known clients, and servers. For clarity, calls to utility functions to enable and disable interrupts, to log statistics (SW Housekeeping) and for abnormal operations are not shown.




6.Science
Obtain Commands
Produce Sci. Data Sets
Post-Proces Data Sets
Ack. Cmds Auto-Restart
Obtain Parameters
7.Uplink
3.Thread Manager
19.Buffer Pool
17.Utility
2.Startup
Cntl. Event Processing
Obtain Data Sets
Analyze and Trigger
Pack Science Dat.
Send Science Dat.
9.Timing
12.Front End Registers
16.Partition Map Builder
19.Buffer Pool
8.Downlink
14.Typer Builder
15.Recoder Builder


7.Uplink
Transfer Cmd Data
Dispatch Messages
19.Buffer Pool
10.DMA
3.Thread Manager
18.Queue Utility


8.Downlink
EA Run Id Packet
Time-Stamp Packets
Transfer Tel. Data
Obtain EA Id
9.Timing
18.Queue Utility
10.DMA
19.Buffer Pool
11.Back End Registers


9.Timing
Manage Mission Time
Manage Intervals
Dispatch Messages
11.Back End Registers
13.Interrrupt Manager
12.Front End Registers
3.Thread Manager

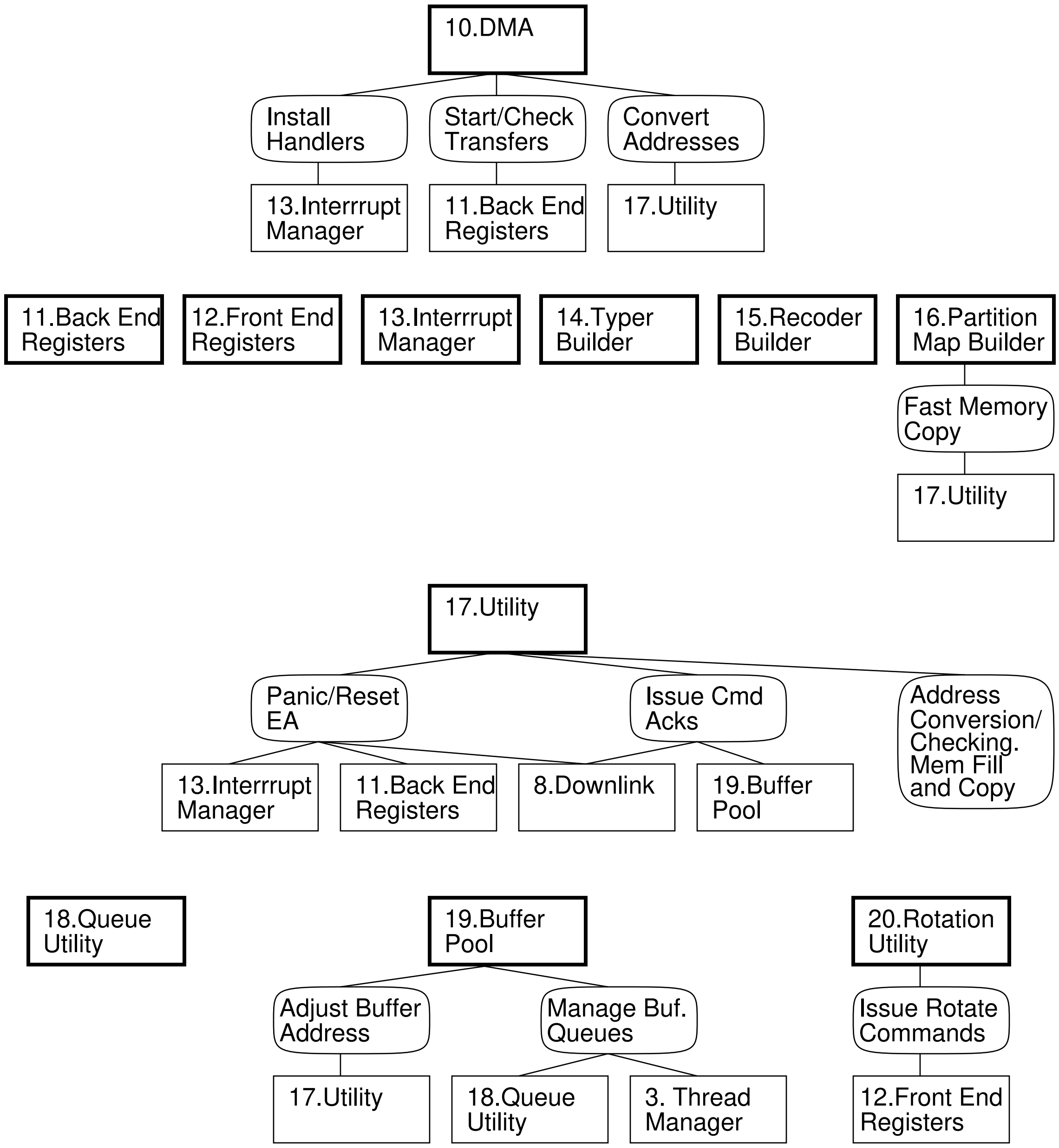


Figure 17 Flight Software Structure Charts

4.3 Module Interfaces

4.3.1 ROM Boot (BOOT)

Purpose

The purpose of the ROM Boot Module is to load and execute code from either the ROM/PRAM Repository, or from the Uplink DMA channel. Since this module must stand alone in Boot ROM, it directly manipulates all hardware, including the DMA controller.

Service	**Description**
Load and start FS	Loads the Flight Software from System Manager ROM and Protected RAM (PRAM) and starts its execution

Known Clients	**Services**	**Reason**
Ground Software	Uplink Boot	Back-door, in case of catastrophe

Required Servers	**Service**	**Reason**
NONE	NONE	NONE

4.3.2 Startup (STARTUP)

Purpose

The purpose of the Startup Module is to install patches loaded by the ROM Boot Module, initialize all of the Flight Software modules, issue a startup message packet, start the Thread Manager, and provide system configuration information for each thread.

Service	Description
Install Patches	Installs patches loaded by the ROM Boot from the PRAM Repository.
Initialize Modules	Calls the initialization routines of all FS modules.
Issue Startup Message	Issues a Startup Message Packet.
Start Multitasking	Transfers control to the Thread Manager's multitasking executive.
Provide Configuration Info.	Supplies each thread with EA specific configuration info.

Known Clients	Services	Reason
All modules	Initialize	Startup calls all initialization funcs.
Science Thread	Config. Info.	Used to obtain configuration info.

Required Servers	Service	Reason
Back End Registers	EA Id	Obtain Event Analyzer Id.
	Watchdog	Start Watchdog Timer.
Utility	Startup Message	Issue Startup Message Packet.
Interrupt Manager	Interrupts	Enable Int's before Multitasking
Thread Manager	Multitasking	Transfer control to start system.

4.3.3 Thread Manager (THREAD)

Purpose

The purpose of the Thread Manager module is to provide the Flight Software with a non-preemptive, multi-tasking executive. This module is responsible for thread creation, scheduling, context switching, and event notification.

Service	**Description**
Thread Creation	Creates control record for a given thread
Multitasking	Manages multiple execution "context"'s
Thread Suspension	Allows threads to wait for events
Event Notification	Notifies waiting threads when events occur.
Thread Existence	Indicates whether or not a given thread has been created

Known Clients	**Services**	**Reason**
All threads	Create	Construct context during init.
	Wait/Notify	Wait for events/notify other threads
Uplink	Notify	Notify thread when cmd ready.
	Exists	Determine if thread exists
Timing	Notify	Notify threads of timing events.
Startup	Multitask	Start thread executions
Buffer Pool	Notify	Notify threads when bufs are avail.

Required Servers	**Service**	**Reason**
Queue Utility	Setup/Enq/Deq	Used for thread scheduling queues.
	Check	Integrity check of crucial structures.
	Element Count	Get number of elements in queue
Interrupt Manager	Wait for Interrupt	When all threads idle, waits for int.
	Enable	Enable interrupts on context switch.
	Disable	Protect context switch operations.

4.3.4 Supervisor Thread (SUPER)

Purpose

The purpose of the Supervisor Thread Module is to provide the ground with the ability to read and write an EA's memory, and to call subroutines within the Flight Software. It also is responible for managing the Watchdog Timer, and for periodically telemetering software statistics to the ground.

Service	**Description**
Read Memory	Allow commanded reads of an EA's memory
Write Memory	Allow commanded writes of an EA's memory
Call Subroutine	Allow commanded code execution
Watchdog Maintainance	Periodically reset the Watchdog's counter.
Software Statistics	Periodically send software statistics to telemetry.

Known Clients	**Services**	**Reason**
Ground Software	Write	Write parameter blocks
	Read	Read contents of EA's memory
	Call	Directly call FS subroutines

Required Servers	**Service**	**Reason**
Thread Manager	Create	Construct thread context
	Wait	Waits for commands/timing events
Uplink	Obtain Cmd Pkts	Gets maintenance commands
Downlink	Access Pkt Data	Supplies ptr to data area of packet
	Send Pkts	Sends statistics packets.
Buffer Pool	Setup pool	Sets up buffer pool for stat. pkts.
	Obtain Stat. Buffers	Obtains tel. buffers for stat. pkts.
	Release Cmd Bufs	Releases command pkt buffers.
Back End Registers	Reset Watchdog	Reset watchdog counter.
Utility	Ack Cmds	Issue command acknowledgments
	Obtain Statistics	Copies and resets SW statistics.
	Log Statistics	Logs SW statistics.
	Fast Copy	Perform fast memory copies.
	Address Validation	Sanity checks command addresses.

4.3.5 Housekeeper Thread (HOUSE)

Purpose

The purpose of the Housekeeper Thread Module is to obtain and telemeter the values of the ASM's engineering sensors.

Service	**Description**
Eng. Sensor Data	Periodically obtain and telemeter ASM eng. sensor data.

Known Clients	**Services**	**Reason**
Ground Software	Get Housekeeping	Obtain ASM Housekeeping Info.

Required Servers	**Service**	**Reason**
Thread Manager	Create	Construct thread context
	Wait	Wait for cmd/rotation/data events
	Notify	Int. client data ready notification
Uplink	Obtain Cmd Pkts	Get Housekeeping commands
Downlink	Access Pkt Data	Supplies ptr to data area of packet.
	Send Pkts	Sends sensor data packets.
Buffer Pool	Setup pool	Sets up buffer pool for sensor pkts.
	Obtain Sensor Bufs	Obtains tel. buffers for sensor pkts.
	Release Cmd Bufs	Releases command pkt buffers.
Front End Registers	Write/Read ASM	Write queries, read replies
Science Thread	Manage ASM IF	Notifies when ASM interface avail.
Timing	Install Minor client	Installs Minortick interrupt client
	Call-back client	Invokes Minortick client function
Utility	Cmd Ack	Issues command acknowledgments
	Fast Copy	Peforms fast memory copies
	Log Statistics	Logs SW statistics.

4.3.6 Science Thread (SCIENCE)

Purpose

The purpose of the Science Thread Module is obtain and process Xray events, detect bursts within the processed data, control ASM rotations, compress processed data, and to telemeter the compressed data.

Service	**Description**
Event Processing	Commanded processing of incoming Xray Events
Automatic EA Restart	Commandable EA restarts at the end of an Observation.

Known Clients	**Services**	**Reason**
Ground Software	Cntl. Science	Stop Science processing
	Cntl. Sci. Telemetry	Start and stop Science telemetry
	EA Restart	Restart EA at end of Observation
Housekeeper Thread	ASM Interface	Notify when ASM Interface avail.

Required Servers	**Service**	**Reason**
Startup	Config. Info	Provides observation configuration
Thread Manager	Create	Construct thread context
	Wait	Wait for cmds/timing/data events
	Notify	Notify data ready/timing events
Uplink	Obtain Cmd Pkts	Get Science Processing commands
Downlink	Access Pkt Data	Supplies ptr to data area of packet
	Send Pkts	Sends dumps and science data.
	Idle indication	Indicates when all packets are sent
Buffer Pool	Setup pool	Sets up buffer pool for science pkts.
	Obtain Science Bufs	Obtains tel. buffers for sci. pkts.
	Release Cmd Bufs	Releases command pkt buffers.
Timing	Install Interval client	Install Interval interrupt client
	Install FirstFlg client	Install First Flag interrupt client
	Invoke Interval client	Invoke Interval Interrupt client
	Invoke FirstFlg client	Invoke First Flag Interrupt client
	Config. BinClock	Configure BinClock and mask
	Schedule Intervals	Schedule Intervals
	Stop Intervals	Stop Interval generation
	Burst HEXTE	Controls HEXTE output burst trig.
Typer Builder	Decode Typer	Unpacks encoded typer tables
	Supply Length	Supplies length of encoded table
	Check Typer	Perform checks on Typer Table
Recoder Builder	Decode Recoder	Unpacks encoded recoder tables
	Supply Length	Supplies length of encoded tables
	Check Recoder	Perform checks on Recoder Table
Partition Map	Supply Length	Supplies length of Partition Map

	Pack Science Data	Pack data using Partition Map
	Check Partition Map	Check integrity of Partition Map
Front End Registers	Read/Write Triggers	Read and write burst trigger signals
	DSP Reset Cntl	Controls DSP's Reset line.
Rotation Utility	Rotation Duration	Provides rotation time of ASM
	Rotate ASM	Forms/issues rotation cmds to ASM
Utility	Cmd Ack	Issues command acknowledgments
	Fast Memory Copy	Performs fast memory copies
	Fast Memory Fill	Performs fast memory fills
	Panic/Reset	Used to auto-restart an EA.

4.3.7 Uplink (UPLINK)

Purpose

The purpose of the Uplink Module is to obtain and deliver uplinked commands.

Service	**Description**
Receive Commands	Obtains and buffers command packets from FDS.
Deliver Commands	Dispatches each command to the addressed thread.

Known Clients	**Services**	**Reason**
All threads	Deliver Cmds	Obtain received commands

Required Servers	**Service**	**Reason**
DMA	Transfer Uplink	Gets pkts from FDS Interface
	Set handler	Install uplink call-back function
	Invoke handler	Invoke uplink call-back function
Buffer Pool	Setup pool	Setup pool of uplink pkt buffers
	Get buffer	Obtain uplink pkt buffer from pool
	Release buffer	Release uplink pkt buffer to pool
	Enqueue buffer	Place uplink buffer into queue
	Dequeue buffer	Remove uplink buffer from queue
	Peek at buffer	Examine first buffer in queue
Queue Utility	Setup Queue	Initialize uplink queue control
	Test for Empty	Test for empty uplink queue
Thread Manager	Exists	Determine if a thread exists
	Notify	Notify thread when cmd is ready
Utility	Log Statistics	Log SW Housekeeping statistics

4.3.8 Downlink (DOWNLINK)

Purpose

The purpose of the Downlink Module is build telemetry packet headers and transfer the packets to the FDS.

Service	**Description**
Build and Transfer Packets	Builds telemetry headers and starts packet transfers.
Panic Packets	Attempts "Panic" transfers before EA resets.
Access Packet Data Area	Provides pointer to a packet's data area (after the headers)
Idle Indication	Indicates when no telemetry packets are being sent or waiting to be sent.

Known Clients	**Services**	**Reason**
All threads	Access Pkt Data	Provides access to pkt data area.
	Transfer Packets	Builds and transfers telemetry pkts.
Utility	Issue panic packets	Issue panic message packets.
	Access Pkt Data	Provides access to pkt data area.
	Transfer Packets	Send Startup/Cmd Ack telemetry
Science Thread	Idle Indication	Wait for observation coda to be sent

Required Servers	**Service**	**Reason**
DMA	Install callback	Installs Downlink Intr. call-back
	Invoke callback	Invokes Downlink Intr. call-back
	Transfer Telemetry	Sends pkts to FDS Interface
	Busy Indicator	Indicates if a transfer is in progress
	Start panic transfer	Sets up system for panic transfer
	End panic transfer	Waits for panic transfer to complete
Buffer Pool	Enqueue buffer	Place telemetry buf into queue
	Dequeue buffer	Remove telemetry buf from queue
	Release buffer	Return telemetry buffer to pool
Queue Utility	Setup Queue	Initialize downlink queue control
	Test for Empty	Tests for an empty downlink queue
Back End Register	Obtain EA Id	Obtains the Event Analyzer Id
Timing	Obtain MET	Time-stamp telemetry packets.
	Obtain Initial MET	Store EA Run Id in telemetry pkts.

4.3.9 Timing (TIMING)

Purpose

The purpose of the Timing Module is to maintain the current "reference" time, and to manage the event processing timing.

Service	**Description**
Handle timing interrupts	Handles and, if necessary, dispatches control based on the MinorTick, Interval, and First Flag interrupts.
Maintain "reference" time	Manages the acquisition of the current MET value.
Provide "initial" time	Provides access to the MET value during which the module was initialized.
Manage event timing	Manage event processing scheduling, interval timing, event time-stamps (binClock duration), and interval specific flags.
Trigger HEXTE	Controls timed assertion/deassertion of the HEXTE output burst trigger signal.

Known Clients	**Services**	**Reason**
All threads	Timing Notifications	Posts MajorTick timing events
Science Thread	Interval client	Installs/invokes Interval client
	First Flag client	Installs/invokes First Flag client
	BinClock Duration	Sets Time-bin duration and mask
	Interval Scheduling	Schedules Intervals
	Stop Intervals	Disables Interval generator/DSP
	Trigger HEXTE	Pulse HEXTE output burst trigger
Housekeeper Thread	MinorTick client	Used for periodic sensor data acq.
Downlink	Time-Stamp	Obtains current time for pkt hdrs.
	Initial Time	Uses this value for EA Run Id.

Required Servers	**Service**	**Reason**
Front End Registers	Test Interrupts	Verify Interval/FF Interrupt latches
	Reset Interrupts	Reset Interval/FF Interrupt latches
	Control Time-bin	Write Bin Clock duration and mask
	Control Interval	Write Interval duration and flags
	Enable Intervals	Enable Interval generation
	Disable Intervals	Disable Interval generation
	Assert DSP Reset	Reset DSP, and Input FIFO
	Cntl. HEXTE Trig.	Assert/Desassert HEXTE trigger
Back End Register	Verify Interrupt	Verify MinorTick Interrupt latch
	Test MajorTick	Test MajorTick pulse latch
	Reset Interrupt/Majr	Reset MinorTick/MajorTick latches
	Read MET	Read current MET value
Interrupt Manager	Set MinorTick hndlr	Install MinorTick Interrupt handler

	Set Interval hndlr	Install Interval Interrupt handler
	Set FirstFlag hndlr	Install First Flag Interrupt handler
	Invoke MinorTick	Invoke MinorTick Int. handler
	Invoke Interval	Invoke Interval Interrupt handler
	Invoke First Flag	Invoke First Flag Interrupt handler
Thread Manager	Notify	Notify threads of MajorTick Pulse

4.3.10 DMA (DMA)

Purpose

The purpose of the DMA Module is to transfer telemetry and uplink packets to and from the FDS interfaces.

Service	**Description**
Transfer Uplink Packets	Programs DMA for uplink packet transfers. Invoke callback, if necessary, when transfer completes.
Transfer Telemetry Packets	Programs DMA for telemetry packet transfers. Invoke callback, if necessary, when transfer completes.
Supply Telemetry in Progress Info	Indicate if a telemetry packet is in the process of being transferred.

Known Clients	**Services**	**Reason**
Uplink	Transfer Uplink	Used to transfer uplink packets.
	Uplink callback	Invoke call-back after transfer
Downlink	Transfer Telemetry	Used to transfer telemetry packets.
	Downlink callback	Invoke call-back after transfer

Required Servers	**Service**	**Reason**
Interrupt Manager	Install uplink	Install Uplink Interrupt handler
	Install downlink	Install Downlink Interrupt handler
	Invoke uplink	Invoke Uplink Interrupt handler
	Invoke downlink	Invoke Downlink Interrupt handler
Back End Registers	Request to Send	Assert FDS Request to Send signal
	Examine latches	Verify uplink/downlink int. latches
	Reset latches	Reset uplink/downlink int. latches
Utility	Pointer conversion	Convert ptr addrs to flat format

4.3.11 Back End Registers (BREG)

Purpose

The purpose of the Back End Registers Module is provide facilities to read and write the CPU Control, Status and Pulse registers, and to obtain the MET register values.

Service	**Description**
Set/Clr bits in Control Register	Selectively sets or clears bits the CPU Control reg.
Read Status Register	Returns the current value of the CPU Status reg.
Write Pulse Register	Writes the contents of the CPU pulse reg.
Read MET Register	Returns current value of the MET register.
Obtain EA Id	Reads EA Id from CPU Status reg. and returns read value.

Known Clients	**Services**	**Reason**
Timing	Read Status	Detect/verify MajorTick/MinorTick
	Write Pulse	Clears Majortick/Minortick latches
	Read MET	Read current MET value.
Startup	EA Id	Obtains EA Id value for config.
	Write Pulse	Start Watchdog Timer
Supervisor Thread	Write Pulse	Reset Watchdog Counter
	Read Status	Indicates Watchdog is running
Downlink	EA Id	Obtains EA Id for App. Id mapping.
DMA	Write Cntl	Assert FDS Request to Send
	Read Status	Check uplink/downlink interrupts
	Write Pulse	Reset uplink/downlink int. latches
Utility	Write Pulse	Start/reset Watchdog during panic.

Required Servers	**Service**	**Reason**
NONE		

4.3.12 Front End Registers (FREG)

Purpose

The purpose of the Front End Registers Module is to provide facilities to read and write the Observation Control, Status and Pulse registers, and to write the BinClock duration, Interval duration registers, and to read and write the ASM Command/Housekeeping registers.

Service	**Description**
Set/Clr bits in Control Register	Selectively sets or clears bits in Obs. Control reg.
Read Status Register	Returns the current value of the Obs. Status reg.
Write Pulse Register	Writes the contents of the Obs. pulse reg.
Write Time-Bin Counter	Writes the duration of the event processing BinClock cntr.
Write Interval Counter	Writes the duration of the event processing interval cntr.
Read/Write ASM Interface Reg.	Writes to and reads from the ASM Cmd/Housekeeping register.

Known Clients	**Services**	**Reason**
Timing	Write Control	Sets Interval specific flags
	Write Control	Sets Time-bin mask
	Write Control	Assert DSP Reset
	Write Control	Pulse HEXTE output trigger
	Write Interval Cnt	Controls interval durations
	Write BinClock Cnt	Controls time-bin durations
	Read Status	Test Interval/FF Interrupts
	Write Pulse	Reset Interval/FF Interrupt latches
Science Thread	Write Control	Reset/run DSP
	Write Control	Trigger Bursts
	Read Status	Detect Bursts
	Write Pulse	Reset burst latches
Housekeeper Thread	Write ASM Reg.	Write query cmds to ASM
	Read ASM Reg.	Read query replies from ASM
Rotation Utility	Write ASM Reg.	Write rotation cmds to ASM
Supervisor Thread	Write ASM Reg.	Writes to ASM control

Required Servers	**Service**	**Reason**
NONE		

4.3.13 Interrupt Manager (INTR)

Purpose

The purpose of the Interrupt Manager Module is to manage the Interrupt Controller and CPU interrupt enable/disable.

Service	Description
Enable Interrupts	Globally enables interrupts on the CPU.
Disable Interrupts	Globally disable interrupts on the CPU.
Nested Interrupt Disables	Provide ability to nest interrupt disables.
Interval Interrupt Call-back	Install/Uninstall Interval Interrupt call-back function.
DMA Interrupt Call-back	Install/Uninstall DMA Interrupt call-back function.
First-Flag Interrupt Call-back	Install/Uninstall First-flag Interrupt call-back function.
MinorTick Interrupt Call-back	Install/Uninstall MinorTick Interrupt call-back function.
Wait for Interrupt	Halt CPU while waiting for an interrupt.

Known Clients	Services	Reason
All modules	Nested Disables	Used during any atomic operation.
Startup	Enable Interrupts	Used before multitasking starts.
Thread Manager	Wait for Interrupt	Used when all threads are waiting.
	Enable	Enable interrupts on context switch.
	Disable	Protect context switching ops.
Timing	Interval Call-back	Used to install interval handlers
	First-Flag Call-back	Used to install data transfer handler
	MinorTick Call-back	Used to manage system time.
DMA	Uplink Call-back	Used to manage uplink
	Downlink Call-back	Used to manage downlink

Required Servers	Service	Reason
NONE		

4.3.14 Typer Builder (TYPER)

Purpose

The purpose of the Typer Builder Module is to generate the Typer Lookup Table contents based on ground specified parameters and to check the integrity of the table. This module assumes that it is invoked by the Science Thread.

Service	**Description**
Construct table contents	Uses parameter blocks to construct Typer Lookup Table.
Provide length of encoded table	Provides the length of the parameter blocks
Check table integrity	Test the integrity of the Typer Lookup Table

Known Clients	**Services**	**Reason**
Science Thread	Construct Table	Sets up table before acquiring data.
	Get Length	Uses length when dumping params.
	Check Integrity	Check table after observation ends

Required Servers	**Service**	**Reason**
Thread Manager	Wait	Allows non-science threads to run

4.3.15 Recoder Builder (RECODER)

Purpose

The purpose of the Recoder Builder Module is to generate the Pulse Height Recoder Lookup Table contents based on ground specified parameters and to check the integrity of the table. This module assumes that it is invoked by the Science Thread.

Service	Description
Construct table contents	Uses parameter blocks to construct Recoder Table.
Provide parameter lengths	Provide the lengths of the specified parameter blocks
Check table integrity	Test the integrity of the Recoder Table.

Known Clients	Services	Reason
Science Thread	Construct Table	Sets up table before acquiring data.
	Get Lengths	Uses lengths when dumping params
	Check Integrity	Check table after observation ends

Required Servers	Service	Reason
Thread Manager	Wait	Allows non-science threads to run

4.3.16 Partition Map (PARTMAP)

Purpose

The purpose of the Partition Map Module is to pack Science Data into telemetry packets, using a specified Partition Map, and to test the integrity of that map.

Service	**Description**
Provide table length	Provide the length of a Partition Map
Verify table contents	Test the integrity of a Partition Map
Pack data	Pack data into buffers using a Partition Map

Known Clients	**Services**	**Reason**
Science Thread	Obtain length	Get length of map for param. dump
	Pack Data	Pack data into telemetry buffers
	Check map	Test integrity of the Partition Map

Required Servers	**Service**	**Reason**
Utility	Fast Memory Copy	Performs fast memory copies

4.3.17 Utility (UTIL)

Purpose

The purpose of the Utility module is to provide the other FS modules with basic CPU dependent utilities. These utilities include fast memory copies, 80286 address conversion and checks, system panic functions, system statistic reporting facilities (i.e. software housekeeping), startup message facilities and command acknowledgment facilities.

Service	Description
Copy Memory	Copies data from one memory location to another.
Fill Memory	Fill memory with a single value.
No Operation Function	Provides a NOP function for short but coarse time delays.
Address Conversion	Convert 80286 addresses to flat formats and visa-versa.
Address Checking	Verify that a 80286 address is within in the 80286 addressable space.
Address Adjustment	Minimize "offset" portion of an 80286 address
Maintain Statistics	Maintains log of Software Housekeeping statistics, and provides access to these values.
Issues Startup Message Packet	Provide mechanism for forming and issuing a Startup Message packet.
Command Acknowledgment	Provide mechanism for forming and issuing command acknowledgment packets.
Issue Panic Messages/Reset EA	Provides mechanism to issue a panic message packet and to reset the EA after the packet has been sent.

Known Clients	Services	Reason
Startup	Startup Msg	Issue Startup Message Packet
All Threads	Cmd Acks	Issue Command Acknowledgments.
All modules	Fast Memory Copy	Performs fast memory copies.
All modules	Fast Memory Fill	Performs fast memory fills.
DMA	Address Conversion	Used to calc. DMA transfer addrs.
All modules	System Panic	Module detects fatal error cond.
All modules	Statistics recording	Record module specific stat's.
Supervisor Thread	Obtain Statistics	Obtains statistics to be telemetered
Supervisor Thread	Test Address	Checks uplinked command args.
Buffer Pool	Adjust Address	Minimizes offset portion of addr.

Required Servers	Service	Reason
Interrupt Manager	Disable Interrupts	Disable interrupts during panics.
Buffer Pool	Setup pool	Setup utility telemetry buffer pool
	Get buffer	Get utility telemetry buffer
Downlink	Access pkt data	Obtain access to data area of pkt
	Send pkt	Send startup and cmd ack pkts

	Panic send	Send panic message packet
Back End Registers	Start Watchdog	Reset EA using Watchdog Timer

4.3.18 Queue Utility (QUEUE)

Purpose

The purpose of the Queue Utility module is to provide the other FS modules with a standard set of routines used to manipulate first-in/first-out lists (queue's).

Service	**Description**
Setup Queue Control	Initializes a client's Queue data structure
Enqueue an element	Adds an element to the end of a queue
Dequeue an element	Removes and returns an element from the front of a queue
Peek at element	Returns a reference to the first element in a queue
Test for empty Queue	Tests to see if a given Queue is empty
Check Queue	Performs consistency checks on a given queue
Element Count	Provide a count of the number of elements in a queue

Known Clients	**Services**	**Reason**
Uplink	Setup/Test	Maintains pending command pkts
Downlink	Setup/Test	Maintains pending telemetry pkts
Buffer Pool	Enq./Deq./Peek	Used to maintain lists of buffers
Thread Manager	Setup/Enq./Deq.	Used for thread scheduling
	Check	Used during thread integrity checks
	Element Count	Used when searching queues

Required Servers	**Service**	**Reason**
NONE	NONE	NONE

4.3.19 Buffer Pool (BUFP)

Purpose

The purpose of the Buffer Pool Module is to provide the other FS modules with services which manipulate contiguous blocks of equally sized memory (i.e. manage pools of data buffers). It also provides services which allow clients to enqueue and dequeue blocks to and from queue's.

Service	**Description**
Setup Pool of Buffers	Initializes a pool of equally sized buffers
Get a buffer	Obtains a buffer from the client's pool
Release a buffer	Releases a buffer to the client's pool
Enqueue a buffer	Enqueue a buffer onto a client specified queue
Dequeue a buffer	Dequeue a buffer from a client specified queue
Peek at buffer	Peek at first buffer in a client specified queue
Notification when buffer released	Notifies specified thread(s) when a buffer is released to an empty buffer pool.

Known Clients	**Services**	**Reason**
Uplink	Setup/Obtain	Initializes and obtains cmd pkt bufs
	Release	Releases obtained cmd pkt bufs
	Enqueue	Places bufs onto uplink queue
	Dequeue	Removes bufs from uplink queue
	Peek	Examines first buf in uplink queue
All threads	Release	Releases consumed cmd pkt bufs
All threads	Setup/Obtain	Initialize and obtain tel. pkt bufs
Downlink	Release	Releases transferred tel. pkt bufs
	Enqueue	Places bufs onto downlink queue
	Dequeue	Removes bufs frm downlink queue
Utility	Setup/Obtain	Initializes for startup/ack/panic bufs

Required Servers	**Service**	**Reason**
Queue Utility	Enq./Deq./Peek	Used to provide queue services
Thread Manager	Notify	Used to provide notification service
Utility	Adjust Address	Minimize offset portion of address

4.3.20 Rotation Utility (ROTUTIL)

Purpose

The purpose of the Rotation Utility Module is to provide ASM rotation time information and to issue rotation commands to the ASM.

Service	**Description**
Compute Rotation Time	Computes rotation times using rotation step counts
Rotate Cameras	Issues rotation commands to the ASM

Known Clients	**Services**	**Reason**
Science Thread (Position Mode)	Rotation Timing	Schedules data acquisition timing.
	Rotate ASM	Rotates ASM according to config.

Required Servers	**Service**	**Reason**
Front End Registers	Write ASM	Write rotation commands

5 Test Provisions

5.1 System Level Testing

The following lists the components needed to perform system level testing on the Flight Software:

Event Analyzer .... Event analyzer hardware is required to run the Flight Software (at least 2 PCA-EA's and at least 1 ASM-EA).

Command Packet Software .... Software is needed which can construct command packets to be delivered to the Flight Software.

Uplink Hardware .... Hardware is needed which implements or simulates the interface to the EDS (System Manager and S/C Simulator).

Telemetry Packet Software.............. Software is needed which can decode telemetry packets generated by the Flight Software.

Telemetry Hardware .... Hardware is needed which implements or simulates the interface to the EA's telemetry hardware (System Manager and S/C Simulator).

Uplink Boot .... A means to set the Uplink Boot flag is required (System Manager and S/C Simulator).

LED Readout .... A means of reading the state of the LED's is required (System Manager and S/C Simulator).

PCA Events....................................... Hardware is required which can deliver simulated PCA events to the PCA Front End hardware's PCA interface.

ASM Interface .... Hardware is required which can deliver ASM events to the ASM Front End interface, and which can read ASM motor commands, and read and reply to ASM sensor queries.

Input Triggers .... A means of controlling the input trigger lines to an EA is required.

Output Triggers................................. A means of reading the state of the Output Trigger lines is required.

Clocks .... Hardware is required to implement or simulate the MicroTick, MinorTick and MajorTick clock signals to an EA (System Manager and S/C Simulator).

MET .... A means of implementing or simulating updates of the MET register is required (System Manager and S/C Simulator).

5.2 Module Level Testing

In order to test each module (unit test), the following hardware is required:

Monitor/Debugger .... The monitor/debugger must read and write memory, execute functions, and read and write CPU registers. A remote symbolic would also be very useful for unit testing.

Logic Analyzer .... The logic analyzer should be capable of tracing and recording the address and data buses of both the CPU and DSP.

Oscilloscope...................................... The scope should be capable of monitoring timing signals down to 1us. If the selected logic analyzer can handle the timing signal rates, a scope may be unneccessary.

6 Requirements Traceability

The requirements satisfied by this document are listed in the **Requirements Traceability Matrix**.

7 Packaging

7.1 Boot ROM

Each Event Analyzer is supplied with its own Boot ROM. Only the ROM Boot Software Module will appear in this ROM. The code in each Boot ROM is identical.

7.2 Subgroup ROM/PRAM Repositories

All final Flight Software shall be supplied in two ROM Repositories, one on each System Manager board for each EA subgroup. The code and data stored in each repository shall be identical. Post-launch changes to the Flight Software shall be uplinked to the subgroup PRAM Repositories. Each EA shall load and install any uplinked patches upon boot.

Appendix A

A.1 Flight Software System Control Flow

This appendix describes the control flow of the Flight Software with respect to the system and major components of the system. This section is intended as a guide, to show the reader how the various modules in the system interact. It is not intended to serve as a set of requirements on each of the modules implementation.

A.1.1 Threads

The Flight Software supports up to 8 different execution threads, under control of the Thread Manager Module. Currently, the Flight Software uses only three of these. They are listed in Section 4.2.1. A thread's state diagram is as follows:

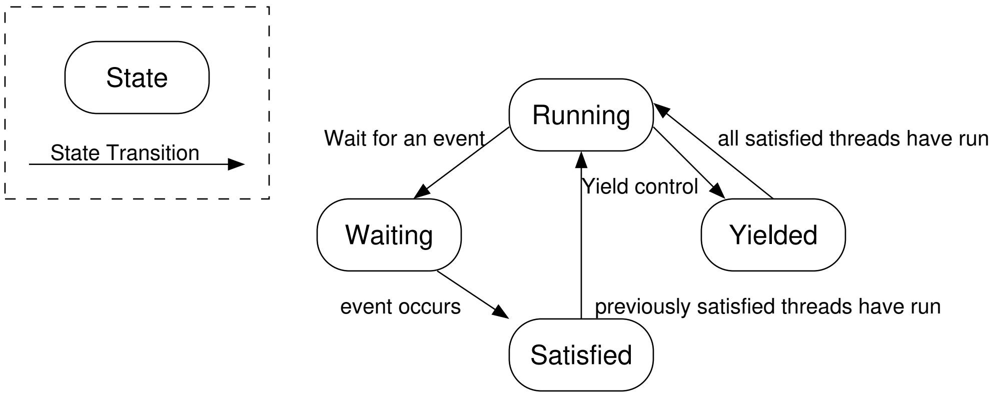


Figure 18 Thread state diagram

The Flight Software uses a non-preemptive, multitasking system. Scheduling occurs whenever the running thread waits for an event or yields control. When waiting for an event, the thread remains inactive until notified that the event has occurred. Once notified, the thread enters a satisfied state, during which it waits until all previously satisfied threads have run and relinquished control. When a thread yields, it enters a yielded state, during which it waits until all satisfied threads and previously yielded threads have run and relinquished control.

A.1.2 Interrupts

The EDS hardware provides the Flight Software with five interrupts. The overall module calling sequence of each of these interrupts is described in the following sections.

A.1.2.1 Interval Interrupt

The Flight Software uses the Interval Interrupt to schedule subsequent intervals, and to measure time with respect to the data processing side of the system. The following shows the overall calling sequence and which modules obtain control during an Interval Interrupt:

```
INTR : Interval Interrupt Handler
        TIMING : Interval Interrupt Client
                SCIENCE : Installed Interval Client
                        TIMING : Interval Scheduling
                                FREGS : Timing Hardware Control
                        THREAD : Event Notification
```

Figure 19 Interval Interrupt Module Calling Sequence

A.1.2.2 Uplink Interrupt

The Flight Software uses the Uplink Interrupt to manage uplink transfers. The following shows the overall calling sequence and which modules obtain control during an Uplink Interrupt:

```
INTR : Uplink Interrupt Handler
        DMA : Uplink Interrupt Client
                UPLINK: Installed Uplink DMA Client
                        QUEUE : Enqueue Element
                        THREAD : Event Notification
                        BUFP : Get buffer
                        DMA : Transfer
```

Figure 20 Uplink Interrupt Module Calling Sequence

A.1.3 Downlink Interrupt

The Flight Software uses the Downlink Interrupt to manage downlink transfers. The following shows the overall calling sequence and which modules obtain control during a Downlink Interrupt:

```
INTR : Downlink Interrupt Handler
        DMA : Downlink Interrupt Client
                DOWNLINK: Installed Downlink DMA Client
                        BUFP : Release Buffer
                        THREAD : Event Notification
                        QUEUE : Dequeue Element
                        DMA : Transfer
```

Figure 21 Downlink Interrupt Module Calling Sequence

A.1.3.1 MinorTick Interrupt

The Flight Software uses the MinorTick Interrupt to manage all time-based operations. The following shows the overall calling sequence and which modules obtain control during a MinorTick Interrupt:

```
INTR : MinorTick Interrupt Handler
        TIMING : MinorTick Interrupt Client
                BREG: Read Register
                BREG: Read MET
                FREGS : Timing Hardware Control
                HOUSEKEEPING : Installed MinorTick Client
                        FREGS : Read/Write ASM IF Register
```

Figure 22 MinorTick Interrupt Module Calling Sequence

A.1.3.2 First-Flag Interrupt

The Flight Software uses the First-Flag Interrupt to find out when the DSP has finished reading events from the previous interval. The following shows the overall calling sequence and which modules obtain control during a First-Flag Interrupt:

```
INTR : First-Flag Interrupt Handler
        TIMING : First-Flag Interrupt Client
                SCIENCE : First-Flag Client
                        THREAD : Event Notification
```

Figure 23 First-Flag Interrupt Module Calling Sequence

Appendix B

B.1 List of TBD Items

This section lists items and issues which need to be resolved:

Document Reference	Section	Resolution Expected by
NONE		

Appendix C

C.1 "C" Compiler Assumptions

The Module and Detailed Designs of the Flight Software make certain general assumptions about the "C" compiler used to build the Flight Software. These assumptions are as follows:

Structure Element Alignment .... The data structures used by the Flight Software assume that the compiler does not align structure elements on boundaries greater than 2 bytes.

Enumerated Types .... The Flight Software uses enumerated types to define sequences of related values, such as thread identifiers. The Flight Software assumes that uninitialized members begin with a value of zero and ascend by 1.

C.2 System Enumerated Types and Constants

The Flight Software makes use of system wide enumerated types and constant values. In some cases, these definitions are compiler dependent values or types, and apply to all modules within the Flight Software. In other cases, the types or values help define the command or telemetry interfaces to the Flight Software, although the scope of a given type or value is closely associated with a single module. Finally, some types and constants are provided at a global level purely for convenience, and are tightly coupled to details in one or two modules (i.e. command packet header sizes and the UPLINK Module).

C.2.1 Constants

The Flight Software defines the following global constant values:

FS_NUM_EA .... (8).................. Number of Event Analyzers in the EDS

FS_DOWNLINK_SIZE .... (1024)............ The maximum size, in bytes, of a Telemetry Packet.

FS_UPLINK_SIZE.......................... (64)................ The maximum size, in bytes, of an Uplink Command Packet.

FS_DOWNLINK_HDR_SIZE .... (16)................ This is the size of the standard Telemetry Packet Header, in bytes, sent at the start of each Flight Software generated Telemetry Packet (see Section 3.2.2).

FS_STREAM_SIZE .... (1008)............ This is the maximum amount of space, in bytes, remaining in a Telemetry Packet for Stream specific data (1024 - 16).

FS_UPLINK_HDR_SIZE .... (10)................ This is the size of the standard Command Packet Header, in bytes, sent at the start of each command interpreted by the Flight Software (see Section 3.2.1).

FS_COMMAND_SIZE .... (54) .... This is the maximum amount of space, in bytes, remaining in a Command Packet for command specific data (64 - 10).

C.2.2 Enumerated Types

The Flight Software defines the following enumerated types:

E_THREAD_ID ...... Thread Identifiers
- THREAD_ID_SUPERVISOR .... Supervisor Thread
- THREAD_ID_SCIENCE .... Science Thread
- THREAD_ID_HOUSEKEEPER .... ASM Housekeeper Thread
- THREAD_ID_ALL_THREADS .... Used to indicate all threads

E_CMD_OPCODE .. Command Operation Codes
- CMD_SUPER_READ_MEM .... Read EA Memory
- CMD_SUPER_WRITE_MEM .... Write EA Memory
- CMD_SUPER_CALL_SUB .... Call EA Subroutine
- CMD_SCIENCE_SUSPEND .... Suspend Science Data Telemetry
- CMD_SCIENCE_RESUME .... Resume Science Data Telemetry
- CMD_SCIENCE_TERMINATE .... Terminate Science Observation
- CMD_SCIENCE_AUTORESTART ... Reset EA at end of Observation

E_CMD_REPLY ..... Command Reply Codes
- CMD_REPLY_OK .... Command executed succesfully
- CMD_REPLY_NONE .... Command does not apply
- CMD_REPLY_UNKNOWN .... Command Opcode not recognized
- CMD_REPLY_ALREADY .... Command's effect already active

E_STREAM_ID ...... Telemetry Packet Stream Identifiers
- STREAM_ID_STARTUP_MSG .... Startup Message
- STREAM_ID_PANIC_MSG .... Panic Message
- STREAM_ID_CMD_ACK .... Command Acknowledgment
- STREAM_ID_SW_HOUSE .... Software Housekeeping
- STREAMID_ASM_SENSOR .... ASM Housekeeping
- STREAM_ID_OBS_PARAM .... Observation Dump
- STREAM_ID_PARTHDR .... Science Partition Header
- STREAM_ID_PARTDATA .... Science Partition Data
- STREAM_ID_OBS_CODA .... Science Observation Coda

E_MSG .... Panic Message Codes
Most message code identifiers have the form, MSG_<module>_<function>_<msg>, where the panic message code

mnemonic identifies the generating module (<module>), function (<function>), and reason (<msg>). For example, MSG_DMA_SEND_BAD_ADDR indicates that a bad transfer address was passed to the DMA Module's "send" function.

E_LOG .... Software Housekeeping Codes
Most housekeeping code names have the form, LOG_<module>_<function>_<item>, where the code mnemonic identifies the generating module (<module>), function (<function>), and (<item>). For example, LOG_UTIL_ACK_DROPPED indicates that value that corresponds to the number of dropped command acknowledgments.

C.2.3 Internal Constants and Type Definitions

The Flight Software uses the following internal global constants and type definitions:

INT8 .... Signed 8-bit value
UINT8 .... Unsigned 8-bit value
INT16 .... Signed 16-bit value
UINT16 .... Unsigned 16-bit value
INT32 .... Signed 32-bit value
UINT32 .... Unsigned 32-bit value
BOOL .... An enumerated type containing:
FALSE = 0
TRUE = 1
LINKABLE .... A structure containing a pointer to another instance of the same structure (used to maintain lists of elements).
NULL .... Constant defined to be a pointer to a void at location 0.

The Flight Software uses "events" or "signals" to control thread scheduling. Each event code must consist of a single bit and must be unique with respect to the other event codes. For this reason, all event codes must be defined globally, to ensure that each is unique. All event codes have the form "EV_<code>" where <code> identifies the specific event.