FFT Mode Module Specification

James Francis

MIT 64-30112.060107

Rev. C

February 28, 1993

Contents

Figures

Tables

1 Introduction

1.1 Overview

This document describes the interface and top level design of the FFT Science Mode. This document relies heavily on the **Science Thread Module Specification**, MIT 64-30112.0601 Rev. E, which provides the document reference list, module perspective, acronym list, module context, interface summaries and thread level structure and function descriptions used by this document.

In this document, Section 1 provides an overview of the operation of FFT Mode. Section 2 describes the mode specific CPU and DSP parameters. Section 3 describes the function callable by the main Science Thread dispatcher. Section 4 describes aspects of the detailed design of FFT Mode, from both the CPU's and DSP's perspective. Section 5 mentions the science requirements addressed by this document, and Section 6 describes the issues involved in testing FFT Mode. Appendix A lists all items which need to be determined or reviewed.

1.2 Theory of Operation

FFT Mode produces three types of data products. These include accumulated Power Density Spectra of accumulated time-series histograms, an accumulated Complex Cross-Spectrum of pairs of these histograms, and a map of the mean values of all accumulated histograms. The number of points in each of the generated spectra is 256.

FFT Mode works by accumulating two time-series histograms over a period of time known as an Accumulation Interval. The two histograms are differentiated by the energy range of the events accumulated into each histogram (see **Recoder Builder Module Specification**). Each histogram consists of a series of "bins", where each bin represents the number of events received during a slice of time during the Accumulation Interval. These bins are known as time-bins, and are formed using the time-stamp within the received processed event words.

After each Accumulation Interval ends, data accumulation stops for a period of time known as a Processing Interval. During this time, FFT Mode computes the mean value of all of the bins in each histogram, records the computed mean values, and subtracts the computed means from each of the bins in each histogram. FFT Mode then computes the Discrete-Time/Discrete-Frequency Complex Fourier Spectra of each of the histograms, using a Fast-Fourier Transform algorithm. It then computes the complex cross-spectrum of the two transforms, and the power spectra of each. The complex cross-spectrum is then added to the spectra computed during previous Processing Intervals, and each power spectrum is added to the power spectra also computed during previous Processing Intervals.

FFT Mode then repeats the process for a configurable number of Accumulation and Processing Intervals, known as a Product Interval, after which the data is packed and transferred out of the EA. The period of time taken to transfer the data is known as a Transfer Interval. FFT Mode repeats the entire process, producing data for each Product Interval, until the observation is terminated.

The following describes this process. At the start of each Accumulation, Processing, or Transfer Interval, the hardware writes an Interval Marker into the DSP's Input FIFO, and resets the time-stamp counter and updates various control and status flags. These flags include:

Sequence Active .... Indicates whether or not processed event words are written into the DSP's Input FIFO. In FFT Mode, this is enabled for Accumulation Intervals, and disabled for Processing and Transfer Intervals.

First Flag .... Indicates whether or not to cause a First-Flag Interrupt on the CPU whenever the DSP reads an Interval Marker from its Input FIFO. This bit appears in the Interval Marker written into the DSP's Input FIFO at the start of the Interval. In FFT Mode, this is only enabled for Transfer Intervals. This bit is used to indicate when a data product is ready to be copied from the DSP's Memory.

Interval Constant .... This two-bit value is or'd with bits 13 and 14 of the processed event word before it is written to the DSP's Input FIFO. In FFT Mode, both bits are always set to zero.

Spillage .... This is a CPU readable status flag, updated by the hardware at the start of each Interval, which indicates whether or not the DSP's Input FIFO filled during the previous Interval.

During a given Accumulation Interval, the Front End hardware looks up the event type, using the Typer Lookup Table, and event pulse height, using the Recoder Lookup Table, and combines the mapped results with the current time-bin counter value to form the processed event word. It then writes the formed word into the DSP's Input FIFO. At the end of each Accumulation Interval, the Front End hardware resets the time-bin counter. During Processing and Transfer Intervals, only Interval Markers are written into the DSP's Input FIFO.

As the Front End hardware forms and writes processed event words and Interval Markers into the DSP's Input FIFO, the DSP reads the Interval Markers and processed event words from the FIFO. When the DSP reads a processed event word, it uses the event word as an address into its histogram buffer, and increments the addressed location by a constant value (see Section 2.2). If the DSP reads an Interval Marker, it stops accumulating the histograms, and proceeds to compute the Mean data products, and the Power Spectra and Cross Spectrum, and adds the computed Power Spectra and Cross Spectrum with those computed during previous Processing Intervals. Once the histogram values are no longer needed, the DSP sets its histogram bins to zero. Once the computations are complete, the DSP polls its Input FIFO until it reads another Interval Marker. If the read Interval Marker has its First Flag bit set to 1, a Transfer Interval is underway and the DSP enters a polling state, waiting for another Interval Marker. Once it reads an Interval Marker which has its First Flag bit set to 0, the Transfer Interval is complete, and the DSP starts constructing the next set of time-series histograms.

In FFT Mode, the CPU is responsible for scheduling the Accumulation, Processing, and Transfer Intervals, and for acquiring and telemetering the data products produced by the DSP. When scheduling the Accumulation and Processing Intervals, the CPU disables First Flag Interrupt generation. When scheduling Transfer Intervals, the CPU enables First Flag Interrupt generation. When the DSP reads the Interval Marker associated with the start of a Transfer Interval, a First Flag Interrupt is generated on the CPU. This interrupt indicates that all computations for the previous Product Interval have completed. The CPU then copies the accumulated data from the DSP memory into its Science Buffer and starts telemetering the copied data.

1.2.1 Data Flow

The following diagram illustrates the overall FFT Mode data flow:

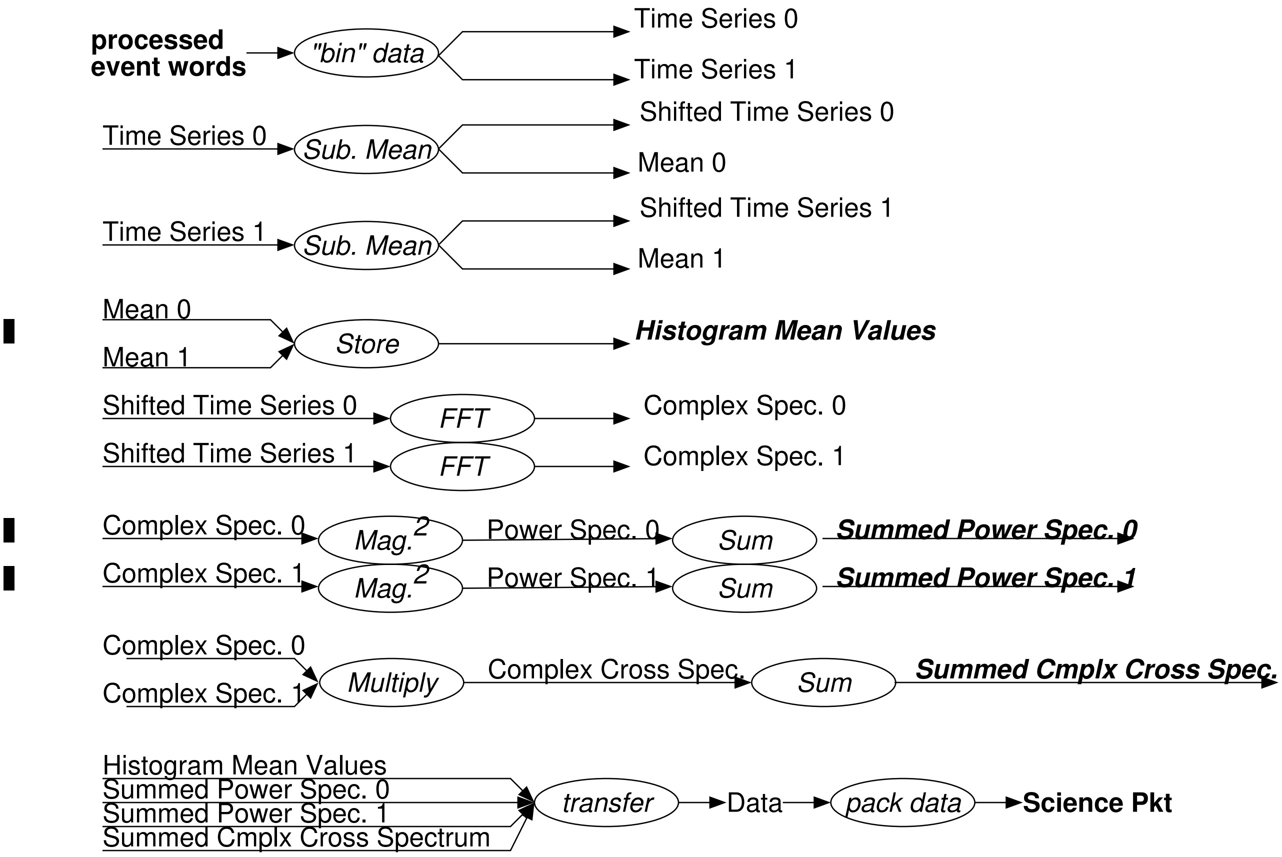


Figure 1 FFT Mode Data Flow

1.3 Data Products

The approximate (error terms are ignored) formulae for the final data products, in terms of the Time-Series histograms are as follows:

Notations

J denotes a constant value

f(n) denotes a real valued function of "n" in the time domain

F(ω) denotes a real valued function of "ω" in the frequency domain

F denotes a complex value of the form, $(F_{real} + jF_{imag})$ where $j = (-1)^{1/2}$

F* is the complex conjugate of **F**, $(F_{real} - jF_{imag})$

Definitions

N	= Number of points in each Time-Series Histogram (256)
W	= Total Number of Product Intervals per Transfer Interval
X	= Bin Scaling Factor. See Section 2.2.
YX^2	= Power Spectra Scaling Factor. See Section 2.2.
ZX^2	= Cross Spectrum Scaling Factor. See Section 2.2.
$a_i(n)$	= Unscaled Time-Series Histogram 0, product interval "i", bin "n"
$b_i(n)$	= Unscaled Time-Series Histogram 1, product interval "i", bin "n"
s_i	= Unscaled Mean Value for Histogram 0, product interval "i"
t_i	= Unscaled Mean Value for Histogram 1, product interval "i"

Output Products

Xs_i	= Scaled Mean Value for Histogram 0, product interval "i"
Xt_i	= Scaled Mean Value for Histogram 1, product interval "i"
P(ω)	= Scaled Summed Power Density Spectrum of $a_i(n)$, i = 0..W
Q(ω)	= Scaled Summed Power Density Spectrum of $b_i(n)$ for i = 0..W
R(ω)	= Scaled Summed Complex Cross Spectrum of $a_i(n)$, $b_i(n)$ for i = 0..W

$Xs_i = 1/N \sum Xa_i(n)$ for "n" = 0 through N-1

$Xt_i = 1/N \sum Xb_i(n)$ for "n" = 0 through N-1

$P(\omega) = YX^2 \sum |\mathbf{A}_i(\omega)|^2$ for all "i" $= Y\sum((XA_{i,real}(\omega))^2 + (XA_{i,imag}(\omega))^2)$ for all "i"

$Q(\omega) = YX^2 \sum |\mathbf{B}_i(\omega)|^2$ for all "i" $= Y\sum((XB_{i,real}(\omega))^2 + (XB_{i,imag}(\omega))^2)$ for all "i"

$\mathbf{R}(\omega) = ZX^2 \sum \mathbf{A}_i(\omega)\mathbf{B}_i(\omega)^*$ for all "i"

$= ZX^2 \sum (A_{i,real}(\omega) + jA_{i,imag}(\omega))(B_{i,real}(\omega) - jB_{i,imag}(\omega))$ for all "i"

$= R_{real}(\omega) + jR_{imag}(\omega)$

where:

$X\mathbf{A}_i(\omega) = \sum (Xa_i(n) - Xs_i)\, e^{-j2\pi\omega n/N}$ for "n" = 0 through N-1

$X\mathbf{B}_i(\omega) = \sum (Xb_i(n) - Xt_i)\, e^{-j2\pi\omega n/N}$ for "n" = 0 through N-1

$R_{real}(\omega) = ZX^2 \sum (A_{i,real}(\omega)B_{i,real}(\omega) + A_{i,imag}(\omega)B_{i,imag}(\omega))$ for all "i"

$R_{imag}(\omega) = ZX^2 \sum (A_{i,imag}(\omega)B_{i,real}(\omega) - A_{i,real}(\omega)B_{i,imag}(\omega))$ for all "i"

1.3.1 DSP

Initially, the DSP's Time-Series and Data Product buffers are set to zero. In FFT Mode, each Product Interval consists of a series of Accumulation and Processing Intervals, followed by a single Transfer Interval. Each type of interval may have a different duration.

During the Accumulation Interval, the DSP generates two time-series histograms. Each element of the histograms represents a time-bin. The histograms are differentiated by pulse-height group or by type classification (see **Recoder Builder Module** and **Typer Builder Module**).

When the DSP reads an Interval Marker, it stops building histograms and starts processing the accumulated data. During the Processing interval, the DSP calculates the mean value of each histogram and stores the results. It then subtracts the means from the corresponding histograms and computes the complex Fourier Transforms of the results, using integer arithmetic. From these complex transformations, it calculates the magnitude squared of each frequency component of each transform and adds the result to the spectra generated in earlier Processing Intervals. In addition to summing the Power Spectra, it performs a complex multiplication between the corresponding frequency components of the pair of Fourier Spectra, and adds the results to the Cross-Spectra generated in earlier Processing Intervals.

After performing these computations, the DSP starts polling its Input FIFO. If the DSP reads an Interval Marker with its First Flag bit set to 1, the DSP assumes that the CPU is copying its data, and polls its Input FIFO until it reads an Interval Marker which has its First Flag bit set to 0. Once the DSP reads an Interval Marker with a First Flag bit set to 0, it proceeds to construct the next set of time-series histograms.

This process continues until the DSP is reset.

1.3.2 CPU

As in every Science Mode, FFT Mode consist of four stages; setup, dump, run, and check. In the setup stage, the CPU side of FFT Mode uses the Science Thread utility functions to initialize the DSP, Partition Map, Typer Table, and the Recoder Table. FFT Mode then uses the **Utility Module** to fill the contents of the DSP's Partition memory with zeros. In addition to these items, FFT Mode uses the **Timing Module** to install FFT Mode specific Interval Interrupt and First-Flag Interrupt client functions, and to initialize the time-bin duration and time-bin nibble mask.

During the dump stage, FFT Mode uses the Science Thread utility functions to dump the contents of the Observation Parameter Block Header, PCA Mode Specific Data, FFT Mode specific data, and all components referenced by these structures, to the ground.

During the run stage, FFT Mode schedules the first Accumulation Interval of the observation, and enables event acquisition. FFT Mode then enters is main processing loop. During this loop, FFT Mode waits for the First-Flag Interrupt client function (see below) to notify the thread that data has been copied into the Science Buffer. Once the data has been copied by the First-Flag Interrupt client function, and telemetry is not suspended via the Observation Parameter Block Header (see **Science Thread Module Specification**, "Observation Parameter Block Header"), or via a command (see below), FFT Mode packs the data into telemetry packets and posts the packets for transfer, using the Science Thread utility function, SCIENCEsend_partdata.

If FFT Mode is configured to govern its telemetry rate via the Observation Parameter Block Header, it waits for at least 1 MajorTick Pulse (1 second) to occur between each science telemetry packet. If FFT Mode is not configured to govern its telemetry usage, it only waits when there are no telemetry packets available to send the data. It does, however, yield control between a certain number of packets, to ensure that the other threads have an opportunity to execute (the number of packets is chosen such that the thread waits/yields at least once per second). Once the entire partition is sent, FFT Mode flags the Science Buffer as ready to take more data, and waits for data to arrive. If science telemetry is suspended, FFT Mode sends only the partition header packets (see **Science Thread Module Specification**, "Partition Header Packet") for each data set acquired.

While obtaining and telemetering science data, FFT Mode periodically (at least once per second) checks for any uplinked commands. If a command arrives, FFT Mode calls the Science Thread utility function, SCIENCEhandle_uplink. This function returns a state variable, which tells FFT Mode to do one of the following: terminate the observation, suspend its telemetry, or resume its telemetry. If FFT Mode is commanded to terminate the observation, it finishes sending the current data set, and enters the observation check stage. If FFT Mode is commanded to suspend its telemetry, it finishes sending the current data set, and continues to accumulate data, but does not send the accumulated data to the ground. If FFT Mode is commanded to resume its telemetry, it finishes accumulating the current data set, and starts sending the data.

When the Accumulation Interval starts, the hardware generates an Interval Interrupt on the CPU. This invokes the Interval Interrupt Client function, which then schedules the Processing Interval. Once the Processing Interval starts, another Interval Interrupt is generated, and the client function schedules the next Accumulation Interval. The Interval Interrupt Client function continues to schedule alternating Accumulation and Processing Intervals until a configurable number of histograms have been processed.

Once the last Processing Interval of the sequence starts, the Interval Interrupt Client function schedules a Transfer Interval, and enables First Flag Interrupt generation. When the DSP reads the Interval Marker, written by the hardware at the start of the Transfer Interval, a First Flag Interrupt is generated on the CPU, and the DSP enters its idle polling loop. When the First Flag Interrupt occurs, FFT Mode's First Flag Interrupt Client function is invoked. The client function then proceeds to copy the accumulated data from the DSP's memory into the CPU's Science Buffer. After the client function copies the data, it zeros the contents of the DSP's data product buffers. The client function then notifies the Science Thread that data is available.

The First-Flag Interrupt client function maintains a flag, in conjunction with the Science Thread, which indicates whether the CPU's Science Buffer is available. When the First-Flag Interrupt client function is invoked, it tests the state of this flag. If the flag indicates that the Science Buffer is "empty," the client function copies the accumulated data into the buffer, and flags the buffer as full. After the Science Thread has finished telemetering the data, it resets the flag to "empty." If, when the client function is invoked, the flag indicates that the Science Buffer is still in use (i.e. FFT Mode hasn't finished telemetering the contents), the First-Flag client function records the occurrence in the Software Housekeeping database, and fills the DSP's data product buffers with zeros without copying the data.

Once the observation is terminated, FFT Mode enters its check stage, where it tests the integrity of the Front End hardware lookup tables and the Partition Map, and forms and telemeters the observation's coda.

2 Data Structures

2.1 FFT Mode Parameter Block

The FFT Mode Parameters immediately follow the PCA Observation Parameter Block as described in the **Science Thread Module Specification**. The mode specific parameters for FFT Mode are as follows:

Accumulation Interval Duration (4 bytes) ... Specifies the number of microticks in an Accumulation Interval. The value is in little-endian format. The ground must ensure that this value is greater than 5ms (unchecked).

Processing Interval Duration (4 bytes) .... Specifies the number of microticks in a Data Processing Interval. The value is in little-endian format. The value of this parameter is a constant, determined by the worst case processing speed of the DSP running FFT Mode. It is provided as a parameter in case the DSP's FFT Mode code is enhanced after launch. In the current implementation, the value of this parameter must be greater than 11011 microticks (10.5 milliseconds).

Transfer Interval Duration (4 bytes) .... Specifies the number of microticks in a Transfer Interval. The value is in little-endian format. The ground must ensure that this value is longer than it takes to transfer a data set from DSP to the science buffer and zero the DSP's memory (see Words/Data Set below).

Product Interval (2 bytes) .... Specifies the number of Accumulation/Processing Intervals per Product Interval minus 1 (i.e. 0 corresponds to 1 Acc/Proc. Interval). The maximum value of this parameter is 7675 (unchecked), and is governed by the amount of memory allocated within the DSP for histogram Mean Values. Since the Transfer Interval Duration has more of an impact on the total dead-time as this parameter decreases, the 50% dead-time requirement may not be met by smaller values of this parameter.

Words/Data Set (2 bytes)............................. Specifies the number of words transferred from the DSP at the end of each Product Interval. The minimum value (unchecked) of this parameter is 1034 words (Means (2) + Cross Spectrum (516) + Power Density Spectra (516)). The maximum value is 16K words (unchecked).

The total percentage of dead-time during an observations is given by:

A - Accumulation Interval Duration
P - Processing Interval Duration
T - Transfer Interval Duration
N - Product Interval

%dead-time = (P + T/N)/(A+P+T/N) * 100

2.2 DSP FFT Mode Parameter Block

The parameters for the DSP FFT Mode are loaded into the DSP via the Auxiliary DSP Load Sequence. This block must be loaded at the 80286 memory location specified by FFT_ADDR_DSP_PARM (see Section 2.4.1). The format of the parameter block is as follows:

Input Scale (1 word) .... This specifies the input scaling factor to the DSP's Time-series histograms. Each accepted event increments the appropriate histogram bin by this amount. If this value is 0, events will not be counted. This value corresponds to "X - Bin Scaling Factor" in Section 1.3.

Output Shift Code (1 word) .... This specifies the output scaling factor for the DSP's Power Density Spectra and Complex Cross Spectrum. This value "controls" the "Y" and "Z" factors shown in Section 1.3. This value can range from 0 to 15 (unchecked) and has the following meanings:

Value	Scale Factor (Y,Z)	Meaning
0	2	Output Scaled up by 2
1	1	Output not scaled
[2..15]	$2^{-(\text{value - 1})}$	Divide Output by powers of 2.

For example, a value of 2 divides the output by 2, a value of 3 divides the output by 4, a value of 4 divides the output by 8 and so on. All divisions are truncated (not rounded) by the DSP.

2.3 DSP Data Accumulation Memory Layout

The DSP uses its Histogram Buffer (see **XTE EDS Hardware Functional Description and Requirements**, "Frontend Core: Table 2- Frontend Memory Map") to accumulate the two time-series histograms, and to store all data products produced during a given Product Interval. This section describes the layout of the DSP's Histogram buffer when performing FFT Mode.

This layout assumes that the Typer and Recoder Tables are configured to produce event words with the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F=1	0	**E**	**T12**	**T11**	**T10**	**T9**	**T8**	**t7**	**t6**	**t5**	**t4**	**t3**	**t2**	**t1**	**t0**

t0-t7	Bits 0-7	Processed Event Time-stamp
T8-T12	Bits 8-12	Unprocessed Event Time-stamp (or dummy energies)
E	Bit 13	Energy Histogram Select
0	Bit 14	This bit must be zero
F=1	Bit 15	First Flag bit, always 1 for event words

In FFT Mode, only the first 256 words in each time-series histogram are processed. This means that only the lower 8 bits of the event word's time-stamp (t0-t7) are used for processed events.

Since, in FFT Mode, only the first 256 words in each time-series histogram are processed, events with a time-stamp value greater than 255 (T8-T12) are ignored by the DSP when computing the Mean values and Fourier spectrum of the histogram. This fact can be used to improve the time-resolution of the algorithm by shortening the time-bin duration while maintaining the Accumulation Interval Duration within its limits. The technique, however, pays the price of extra dead-time while reading and processing events which occur later in the Interval.

Figure 2 FFT Mode Event Word Format

Using the event word format, shown in Figure 2, the DSP's Histogram Buffer is organized as follows:

Section	Component(s)	Element(s)	Buffer Address
Mean Values (2 groups/acc. @ 1 word/value) Size = 15352 words	Acc. Interval 7675	Energy Group 1	0x07FFF
		Energy Group 0	0x07FFE
	Acc. Int.'s 2 - 7674	Energy Groups 0 and 1	0x0440C - 0x07FFD
	Acc. Interval 1	Energy Group 1	0x0440B
		Energy Group 0	0x0440A
	Acc. Interval 0	Energy Group 1	0x04409
		Energy Group 0	0x04408
Summed Cross Spectrum (2 words/part @ 2 parts/freq.) Size = 516words	Freq. Index 128	Imaginary Part	0x04406 - 0x04407
		Real Part	0x04404 - 0x04405
	Freq. Indices 2-127	Real and Imaginary Parts	0x0420C - 0x04403
	Freq. Index 1	Imaginary Part	0x0420A - 0x0420B
		Real Part	0x04208 - 0x04209
	Freq. Index 0	Imaginary part	0x04206 - 0x04207
		Real part	0x04204 - 0x04205
Summed Power Density Spectra (2 words/freq.) Size = 516words	Energy Group 1 Size = 258words	Freq. Index 128	0x04202 - 0x04203
		Freq. Indices 2 - 127	0x04106 - 0x04201
		Freq. Index 1	0x04104 - 0x04105
		Freq. Index 0	0x04102 - 0x04103
	Energy Group 0 Size = 258words	Freq. Index 128	0x04100 - 0x04101
		Freq. Indices 2 - 127	0x04004 - 0x040FF
		Freq. Index 1	0x04002 - 0x04003
		Freq. Index 0	0x04000 - 0x04001
Time-series (1 word/bin) Size = 16Kwords This section is automatically addressed by the processed event words read from the DSP's Input FIFO.	Energy Group 1 Size = 8Kwords	*Bins 256 - 8191*	*0x02100 - 0x03FFF*
		Time-bin 255	0x020FF
		Time-bins 2 - 254	0x02002 - 0x020FE
		Time-bin 1	0x02001
		Time-bin 0	0x02000
	Energy Group 0 Size = 8Kwords	*Bins 256 - 8191*	*0x00100 - 0x01FFF*
		Time-bin 255	0x000FF
		Time-bins 2 - 254	0x00002 - 0x000FE
		Time-bin 1	0x00001
		Time-bin 0	0x00000

Table 1 DSP Histgram Buffer Layout

2.4 Constants

2.4.1 FFT_ADDR_DSP_PARM (0x068000 flat-address)

This specifies the 80286 address of the shared DSP memory location which contains the DSP Parameter Block for FFT Mode (see Section 2.2). This location maps to the start of the DSP Scratch Data RAM (i.e. DSP data address 0x04000).

2.4.2 FFT_IC (0)

This constant defines the interval constant value used by FFT Mode.

2.4.3 FFT_OUTSTANDING_FF (2L)

This constant defines the number of outstanding First Flag Interrupts before the Interval Interrupt client function assumes that the DSP is hung.

2.4.4 FFT_PARTNUM_ADJUST (1L)

This constant is subtracted from the internal partition counter to form the partition number as seen in the Partition Header telemetry packets.

2.4.5 LOG_SCIMODE_FF_DROP_PART (SCIENCE)

The First Flag client function, fft_firstflag(), increments the Software Housekeeping statistic whenever it cannot copy a partition of data because the requisite amount of buffer space is not available (i.e. telemetry backlogged science buffer). Refer to the **Science Thread Module Specification**, "Constants; LOG_SCIMODE_FF_DROP_PART" for the general meaning and form of this statistic.

2.4.6 MSG_SCIMODE_RUN_MODE_NULL_HDR (SCIENCE)

FFTmode() emits this panic message if it is called with a NULL parameter block pointer. Refer to the **Science Thread Module Specification**, "Constants; MSG_SCIMODE_RUN_MODE_NULL_HDR" for the general meaning and form of this panic message.

2.4.7 MSG_SCIMODE_PROCESS_UNLOCKED_BUF (SCIENCE)

fft_process() invokes this panic message if it fails to obtain a filled science buffer section (i.e. the First Flag client function failed to copy data into the buffer or enqueue the buffer). See the **Science Thread Module Specification**, "Constants; MSG_SCIMODE_PROCESS_UNLOCKED_BUF" for the general meaning and form of this panic message.

2.4.8 MSG_SCIMODE_PROCESS_BAD_PARTNUM (SCIENCE)

fft_process() invokes this panic message if it is invoked with an invalid partition number. See the **Science Thread Module Specification**, "Constants; MSG_SCIMODE_PROCESS_BAD_PARTNUM" for the general meaning and form of this panic message.

2.4.9 MSG_SCIMODE_IV_DEAD_DSP (SCIENCE)

The Interval Interrupt client function, fft_interval() emits this panic message whenever it suspects that the DSP is hung. Single Bit Mode passes the partition counter as the argument to the panic message. Refer to the **Science Thread Module Specification**, "Constants; MSG_SCIMODE_IV_DEAD_DSP" for the general meaning and form of this panic message.

2.4.10 MSG_FFT_IV_BAD_IVTYPE (FFT)

The Interval Interrupt client function, fft_interval() emits this panic message whenever its internal interval type value is invalid. The message should read something like:

"PANIC: -- fft_interval([%d]) -- Invalid Interval Type",
where %d specifies offending value.

3 Functions

3.1 FFTrun_mode() - Runs a FFT Mode Observation

Usage

void FFTrun_mode (UINT8 **hdr*);

Description

FFTrun_mode() performs a FFT Mode observation. "*hdr*" points to the Observation Parameter Block for the observation. The FFT Mode Parameters must immediately follow the block.

This function does not return until the observation is terminated.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) Since this function waits and yields, it must be run from a thread.

See Also

Science Thread Module Specification

4 Detailed Design Considerations

4.1 Data Flow

The following illustrates the overall data flow for the FFT Science Mode:

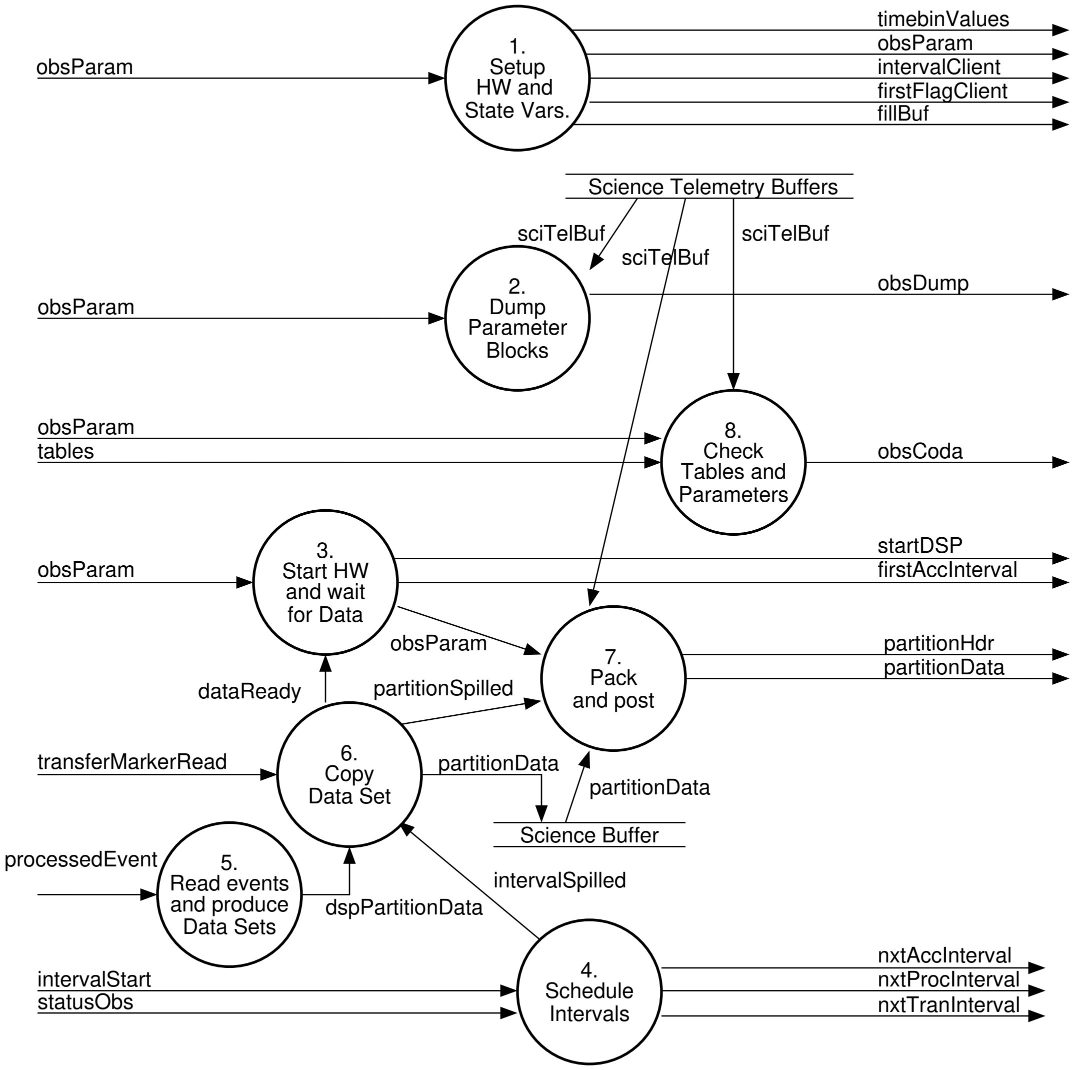


Figure 3 FFT Science Mode Data Flow

To keep the diagram as simple as possible, Figure 3 does not show any command handling, error checking or panic messages.

The first operation performed by FFT Mode (bubble 1) is to setup the system to perform an observation. It configures Front End hardware to run the observation, and installs the FFT Mode's Interval Interrupt Client (intervalClient) and First Flag Interrupt Client (firstFlagClient) functions (the operation of these client functions are represented by process bubbles 4 and 6, respectively). FFT Mode configures most of the Front End hardware using utility functions provided by the main **Science Thread Module**. These functions use the Observation Parameter Block (obsParam) to obtain the required parameters. In addition to these utility functions, FFT Mode sets up the Front End's timebin duration and mask (timebinValues), using the **Timing Module**, and fills the contents of the DSP's Histgoram memory using the **Utility Module** (fillBuf).

Once the hardware has been configured, and the client functions have been installed, FFT Mode dumps the Observation Parameters (obsDump) to the ground (bubble 2), using telemetry buffers provided by the Science Telemetry Buffer Pool (sciTelBuf).

Once the parameter dump is complete, FFT Mode is ready to run the observation (bubble 3). It starts by releasing the DSP's reset line (startDSP) and scheduling the first Accumulation Interval of the observation (firstAccInterval). It then waits for completed data sets.

As the timing circuitry generates interval interrupts (intervalStart) at the start of each Accumulation, Processing or Transfer Interval, the installed Interval Interrupt client function is called (bubble 4). When invoked at the start of each Accumulation Interval, the client function schedules the subsequent Processing Interval (nxtProcInterval), which starts upon the completion of the current Accumulation Interval. When invoked at the start of each Processing Interval, this function reads the Observation Status Register (statusObs) to determine if the DSP's Input FIFO spilled during the preceeding Accumulation Interval (intervalSpilled). This flag is set by any of the Accumulation Intervals within the Product Interval. Once the Processing Interval completes, the Interval Interrupt client function then schedules either another Accumulation Interval (nxtAccInterval), or a Transfer Interval (nxtTranInterval), depending on whether or not the configured number of Accumulation/Processing Intervals have completed. The client function only enables First Flag Interrupt generation when scheduling Transfer Intervals.

Meanwhile, the DSP reads incoming processed event words (processedEvent), and generates the data products (bubble 5) in the DSP's memory. At the start of each Accumulation, Processing or Transfer Interval, the hardware writes an Interval Marker into the DSP's Input FIFO. Interval Markers written at the start of Accumulation or Processing Intervals have their respective First Flag bits set to 0. Interval Markers written at the start of Transfer Intervals have their

First Flag bits set to 1. When the DSP reads a Transfer Interval marker (transferMarkerRead), a First Flag Interrupt is generated, which, in turn, calls the First Flag Interrupt client function (bubble 6). The First Flag Interrupt indicates that the Transfer Interval has started, and that the data set produced during the previous series of Accumulation and Processing Intervals (i.e. the previous Product Interval) is complete. The First Flag Interrupt Client function copies completed data (dspPartitionData) from the DSP's external RAM to the CPU's Science Buffer (partitionData). This function then fills the DSP's data product area (not shown) with zeros, and notifies FFT Mode that data is read to be telemetered (dataReady). The First Flag Interrupt client function also copies the state of the spillage flag set by the Interval Interrupt Client function during the previous series of Accumulation Intervals (intervalSpilled), to a flag associated with the copied data (partitionSpilled). FFT Mode then packs the data into telemetry buffers (sciTelBuf), and posts the packets (partitionHdr, partitionData).

Once the observation is terminated, FFT Mode checks the Front End hardware tables (tables) against the original parameter set (obsParam) and issues a report (obsCoda) using a science telemetry buffer (sciTelBuf).

4.2 Control Considerations

4.2.1 Execution States

When FFT Mode runs an observation, five "concurrent" processes are involved; hardware event processing, DSP event and histogram processing, Interval Interrupt processing, First Flag Interrupt processing, and partition telemetry processing (interrupt handling can be viewed as a form of concurrency). Each of these processes can be viewed as finite state machines, oriented around the data produced during a given interval. The states for each process are as follows:

4.2.1.1 Hardware Event Processing States

Disabled .... Disabled
No interval markers or events are written to the DSP's Input FIFO.

Start A/X .... Start Interval A of Product Interval X.
Start Interval A (either Accumulation, Processing or Transfer Interval) of Product Interval X, reset the timebin counter and write an Interval Marker into the DSP's Input FIFO. If the Interval is flagged as the first (always an Accumulation Interval), synchronize the start of the Interval to the next MajorTick boundary. If not, start the Interval immediately as the previous Interval completes. Accumulation Intervals have data accumulation enabled, and First Flag Interrupt generation disabled. Processing Intervals have both data accumulation and First Flag Interrupt generation

disabled. Transfer Intervals have data acccumulation disabled, but First Flag Interrupt generation enabled.

Process A/X .... Process Accumulation Interval A of Product Interval X.
Time-stamp and process events, and write the processed events into the DSP's Input FIFO.

4.2.1.2 DSP Event Processing States

Reset .... Reset
The DSP is held in a reset state.

Initialize .... Initialize
The DSP starts running, performing any required initialization procedures.

Start A/X .... Process Interval Marker A of Product Interval X
Read Interval Marker, written by the hardware at the start of either an Accumulation, Processing or Transfer Interval A, from the Input FIFO. Whenever an Interval Marker is read which has its First Flag bit set to 0, the DSP switches back and forth between Accumulation and Processing operations. Whenever an Interval Marker is read which has its First Flag bit set to 1, the DSP stops accumulation and processing, and polls its Input FIFO until another Interval Marker is read. When the Interval Marker is read with a First Flag bit set to 1, a First Flag Interrupt is generated on the CPU.

Histogram A/X .... Process Events forming the Time-series histograms
Read events, written by the hardware during Accumulation Interval A of Product Interval X, from the Input FIFO, and generate the Time-series histograms.

Process A/X .... Process the Time-series histograms to form data sets.
Compute the Means, Power spectra and Cross spectrum of the previously built histgrorams. Store the mean values, and add the Power spectra and Cross spectrum to values computed during previous Processing Intervals. After the computations are complete, the DSP fills the histogram bins with zeros.

BusyWait A/X .... Poll Input FIFO for next Interval Marker
Continue to read the Input FIFO until another Interval Marker is read, discarding any read event words.

4.2.1.3 Interval Interrupt Processing States

Wait A/X .... Wait for Interval Interrupt
Wait for the Interval Interrupt which indicates that Accumulation, Processing or Transfer Interval A has started.

Accumulation A/X .... Schedule Accumulation Interval A of Product Interval X
Whenever a Processing or Transfer Interval starts, it causes an Interval Interrupt. Except when the Processing Interval is the last of a Product Interval, the Interval Interrupt Client function then

schedules Accumulation Interval A, enables data acquisition and disables First Flag Interrupt generation.

Processing P/X .... Schedule Processing Interval P of Product Interval X.
Whenever an Accumulation Interval starts, it causes an Interval Interrupt. The Interval Interrupt Client function then schedules Processing Interval P, and disables data acquisition and First Flag Interrupt generation.

Transfer T/X .... Schedule Transfer Interval T of Product Interval X.
Whenever the last Processing Interval of a Product Interval starts, it causes an Interval Interrupt. The Interval Interrupt Client function then schedules Transfer Interval T, disables data acquisition and enables First Flag Interrupt generation.

4.2.1.4 First Flag Interrupt Processing States

Wait X .... Wait for First Flag Interrupt from Product Interval X.
Wait for the First Flag Interrupt caused by the DSP reading the Interval Marker that was written at the start of the Transfer Interval for Product Interval X. This indicates that the DSP has completed all processing during the previous series of Accumulation and Processing Intervals.

Process X .... Process data set from Product Interval X.
When the DSP reads the Interval Marker written at the start of the Transfer Interval of Product Interval X, the hardware generates a First Flag Interrupt. The First Flag Interrupt Client function then copies the data sets produced during the previous series of Accumulation and Processing Intervals, and fills the data set buffers with zeros.

4.2.1.5 Telemetry Processing States

Start DSP .... Start DSP Execution
When starting the observation, FFT Mode releases the DSP's reset line and waits at least one MajorTick, so that the DSP has time to complete its initialization procedure.
NOTE: Since TIMINGfirst_interval() guarantees at least 1 MajorTick before starting the first interval, no special action on the part of FFT Mode is required.

Accumulation 0/0 .... Schedule first Accumulation Interval.
To start the observation, FFT Mode schedules the first Accumulation Interval. The Interval will start on the next MajorTick boundary which is an even multiple of 16 of the MET value.

Wait X .... Wait for Data Set X
Once started, FFT Mode waits for a data set, produced by events which occurred during Product Interval X, to be completed and copied from the DSP.

Process X .... Process Data Set X
Once the data set, produced by events which occurred during Product Interval X, is ready, FFT Mode packs and telemeters the data set.

Terminate .... Terminate Observation
When commanded to terminate an observation, FFT Mode finishes telemetering the current data set, and then terminates the observation.

4.2.2 Command Handling

FFT Mode ensures that commands are handled during event processing, such that other threads are not held up due to pending science command packets. Whenever the thread waits for an event, it also waits for new uplink. Whenever the thread yields, it always checks for pending uplink before issuing the yield call. These two conventions ensure that the thread always handles pending commands before other threads obtain control.

4.3 Interval Timing Description

This section describes the timing issues involved when running FFT Mode. This section assumes that there are two Accumulation and Processing Intervals for each Product Interval. This section relies on the states of the various processes, described in Sections 4.2.1.1, 4.2.1.2, 4.2.1.3, 4.2.1.4, and 4.2.1.5.

Time	Thread	HW	DSP	Interval Int.	First Flag Int.
t0		Disabled	Reset	Wait 0/0	Wait 0
t1	Start DSP				
t2			Initialize		
t3	Accumulation 0/0				
t4	Wait 0	Start 0/0			
t5		Process 0/0	Start 0/0	Processing 1/0	
t6			Histogram 0/0		
t7		Start 1/0			
t8			Start 1/0	Accumulation 2/0	
t9			Process 1/0		
t10		Start 2/0			
t11		Process 2/0	Start 2/0	Processing 3/0	
t12			Histogram 2/0		
t13		Start 3/0			
t14			Start 3/0	Transfer 4/0	
t15			Process 3/0		
t16		Start 4/0			
t17			Start 4/0	Accumulation 0/1	
t18			BusyWait 4/0		Process 0
t19	Process 0	Start 0/1			Wait 1
t20	Wait 1	Process 0/1	Start 0/1	Processing 1/1	
t21			Histogram 0/1		
t22		Start 1/1			
t23			Start 1/1	Accumulation 2/1	
t24			Process 1/1		
t25		Start 2/1			
t26		Process 2/1	Start 2/1	Processing 3/1	
t27			Histogram 2/1		
t28		Start 3/1			
t29			Start 3/1	Transfer 4/1	
t30			Process 3/1		
t31		Start 4/1			
t32			Start 4/1	Accumulation 0/2	
t33			BusyWait 4/1		Process 1
t34	Process 1	Start 0/2			Wait 2
t35	Wait 2	Process 0/2	Start 0/2	Processing 1/2	
t36	Terminate		Histogram 0/2		
t37		Disabled	Reset		

Table 2 FFT Mode Process States

FFT Mode starts with the hardware in a disabled state, the DSP in a reset state, the Interval Interrupt Client function installed and waiting for the first Accumulation Interval to occur, and the First Flag Interrupt Client function installed and waiting for the first Product Interval to complete (t0). The thread starts the DSP by releasing its reset line (t1), and then waits for 1 MajorTick (via TIMING-first_interval()) for the DSP to perform its initialization procedure (t2). All DSP initialization procedures shall take less than one MajorTick to execute. Once the DSP has performed its initialization procedure, the thread schedules first Accumulation Interval 0/0 (t3), and waits for the first data set to arrive (t4).

Once the first Accumulation Interval 0/0 starts (t4), the hardware writes an Interval Marker into the DSP's Input FIFO, and generates an Interval Interrupt on the CPU. The hardware then proceeds to time-stamp and write event words into the DSP's Input FIFO (t5).

Once the Interval starts, the Interval Interrupt Client function then schedules a Processing Interval 1/0 (t5). The Interval Client function must always finish before the current Accumulation Interval ends (t7). At some point before the running Accumulation Interval ends (t7), the DSP reads the Interval Marker (t5) and processes the events which occur during Accumulation Interval 0/0, generating the two time-series histograms (t6). Although the DSP must always read the marker before the Accumulation Interval ends, it may continue to process the Interval's events during part of the Processing Interval.

When Processing Interval 1/0 starts (and Accumulation Interval 0/0 ends) (t7), the hardware writes another Interval Marker into the DSP's Input FIFO, and generates an Interval Interrupt. The Interval Interrupt Client function then schedules next Accumulation Interval 2/0 (t8). The Interval Interrupt Client function continues this process of alternating Accumulation and Processing Intervals until the last Processing Interval of the Product Interval starts (t8 - t13). When the last Processing Interval of the Product Interval starts (t13), the Interval Interrupt schedules Transfer Interval 4/0 (t14) and enables First Flag Interrupt generation. Once a Transfer Interval starts (t16), the client function schedules the first Accumulation Interval 0/1 of the next Product Interval (t17).

When the DSP reads the Interval Marker written at the start of the Processing Interval (t8), it stops building the histograms and proceeds to compute the mean values, power spectra, and cross-spectrum of the accumulated histograms (t9). This processing must complete before the end of the Processing Interval (t10). Once this processing is complete (not shown), the DSP polls its Input FIFO until another Interval Marker is read (t11). After the DSP reads the Interval Marker, it accumulates the next set of histograms (t12). The DSP continues to alternate between accumulating histograms and computing data products until it reads an Interval Marker which has its First Flag bit set to 1 (t13 - t17). When the DSP reads an Interval Marker which has its First Flag bit set to 1 (t17), the DSP enters a polling state, where it reads its Input FIFO until it obtains another Interval

Marker (t20). Once the Interval Marker has been read, the DSP proceeds to accumulate the next set of histograms. The DSP continues this process of alternating between a series of histogram generation and data processing, and polling until it is reset (t20 - t37).

When the DSP reads an Interval Marker (t17), written by the hardware at the start of a Transfer Interval (t16), which has its First Flag bit set, a First Flag Interrupt is generated on the CPU (t18). The First Flag Interrupt Client function then copies the accumulated data sets from the DSP's memory into the Science Buffer, zeros the DSP's area which was copied, and notifies the Science Thread that data is ready. The copying and zeroing operation must complete before the DSP starts its next processing operation (t24).

When the thread is notified that data is ready, it packs and telemeters the data from Product Interval 0 (t19). This operation must complete before the First Flag Interrupt Client function attempts to obtain the data from next Product Interval (t33). Once the thread finishes processing the data, it waits again for more data (t20).

This process continues (t20 - t35) until the thread receives a terminate command (t36). When a terminate command is received, the thread finishes sending whatever data it was sending at the time, shuts down the hardware and DSP, and proceeds to execute the "check" stage of the observation (t37).

4.4 Overthroughput Descriptions

FFT Mode attempts to detect, and gracefully handle several overthroughput conditions. The following sections describe each condition, whether or not FFT Mode handles the condition, and if so, how it's handled.

4.4.1 Event Rate exceeds DSP Rate

If the sustained input event rate, delivered by the hardware, exceeds the rate at which the DSP can read event words from its Input FIFO, during a given Accumulation Interval, the DSP's Input FIFO will fill. If the FIFO becomes completely full, the hardware stops writing events to the FIFO for the remainder of the Accumulation Interval. At the start of the Processing Interval, the hardware sets a CPU readable "spillage" flag.

Whenever the Interval Interrupt client function is invoked by the start of a Processing Interval, the client function reads the CPU readable spillage flag. The Interval Interrupt Client function maintains this flag until the end of the Product Interval. The client function then passes this flag on to the First Flag Interrupt Client function, which then passes the flag to the thread. The thread then indicates the spillage condition in the Partition Header Packet. Spillage during any Accumulation Interval in a Product Interval is indicated by the spillage flag being set for the entire data set of that Product Interval.

4.4.2 DSP Accumulation Rate causes "Bin" or Product overflow

If the event rate into a particular histogram bin is too high, there is a possiblity that the mean subtracted bin value will overflow, the intermediate results during the FFT will overflow, or the bin's count will wrap. Typically, these conditions arise from an improperly configured observation. FFT Mode provides no explicit mechanisms for detecting these conditions. It is up to the ground to configure the observation such that these conditions never occur. Since each histogram is mean-subtracted, the power-spectra at frequency 0 should be close to zero. If results at frequency 0 are large, it indicates a possible overflow problem or computational error. Other than that, it is difficult to detect these conditions from the telemetered data, since such errors tend to be smeared across the spectrum during the FFT computation. Ground detection of the problem is further compounded by the fact that each Accumulation/Processing Interval spectra is summed with the previous spectra accumulated during the Product Interval. Analysis of data from other modes on other EA's may provide a better indication of when this condition occurs.

Another form of overflow can happen if the summing of the 32-bit Power Density Spectra or Complex Cross-Spectrum values overflows. This can occur if the number of Product Intervals is too high relative to the input event rate and distribution, input scaling factor and output scaling factors.

4.4.3 Processing Interval Rate exceeds DSP Processing Time

If the Processing Interval duration is too short, the DSP will not have enough time to complete its spectral computations. If the next Interval is an Accumulation Interval, event words can pile up in the DSP's Input FIFO, causing spillage, as described in Section 4.4.1. If the next Interval is a Transfer Interval, this condition will cause the First Flag Interrupt to be delayed, shortening the available data transfer time and possibly causing data corruption, or worse, a system panic (see Section 4.4.4). This condition arises due to an incorrectly configured observation, or corruption of the observation's parameters.

4.4.4 Data Set Rate exceeds First Flag Copy Rate

If the Transfer Intervals are scheduled short enough, and the data set size is set large enough, the First Flag Interrupt Client function does not have enough time to copy data from the DSP and zero the copied memory. In the case where the transfer time overlaps with Accumulation Intervals, the system will continue to operate correctly, since the DSP will be accumulating data in a separate memory area. In the case where the transfer time overlaps Processing Intervals, outright corruption of this and subsequent data sets will occur. If the transfer time overlaps with the next Transfer Interval, a second First Flag Interrupt request will occur before the completion of the first. In this case, the condition is detected and handled by the **Timing Module** when the First Flag Interrupt Client function returns. When the **Timing Module** detects this occurrence, it will issue a panic and reset the EA. Overloaded First Flag Interrupt processing can happen due to one of two causes; the observation is improperly configured, or the system has been corrupted.

4.4.5 First Flag Copy Rate exceeds Data Packing Rate

If the observation is configured such that the rate at which data sets are copied by the First Flag Interrupt Client function is faster than the thread can pack the data into telemetry packets, the First Flag Interrupt Client function will drop complete data sets. The ground can detect this condition by noticing gaps in the Partition Numbers, contained in the Partition Header Packets. This occurence is also recorded in the Software Housekeeping statistics.

4.4.6 Data Packing Rate exceeds Telemetry Rate

If the thread is packing data into telemetry packets faster than they are transferred out of the EA (due to either governing, or due to SDS backlog), the system will run out of telemetry packets. This backlogs the packing process, which then causes condition 4.4.5.

4.4.7 Dropped Data Set Implementation Notes

Data sets are dropped whenever the First Flag Interrupt Client function has nowhere to copy a data set. This system is implemented using a flag between the thread and the First Flag Interrupt Client function. When the First Flag Interrupt Client function copies data into the Science Buffer, it sets a flag indicating that the buffer is in use. When the thread finishes with the buffer, it clears this flag. If, while the thread is processing the buffer, another First Flag Interrupt occurs, the First Flag Interrupt Client function checks the state of the flag, and if it indicates that the Science Buffer is still in use, the function discards the data set, updates any bookkeeping variables, and fills the copied DSP memory.

4.5 Module Structure

4.5.1 FFT Mode CPU Function Structure

The following diagram illustrates the overall structure of the FFT Science Mode. Bolded boxes with bolded text represent functions provided by modules other than the **Science Module** or FFT Mode. Bolded boxes with unbolded text represent utility functions provided by the **Science Thread Module**. All other boxes represent functions which are private to FFT Mode.

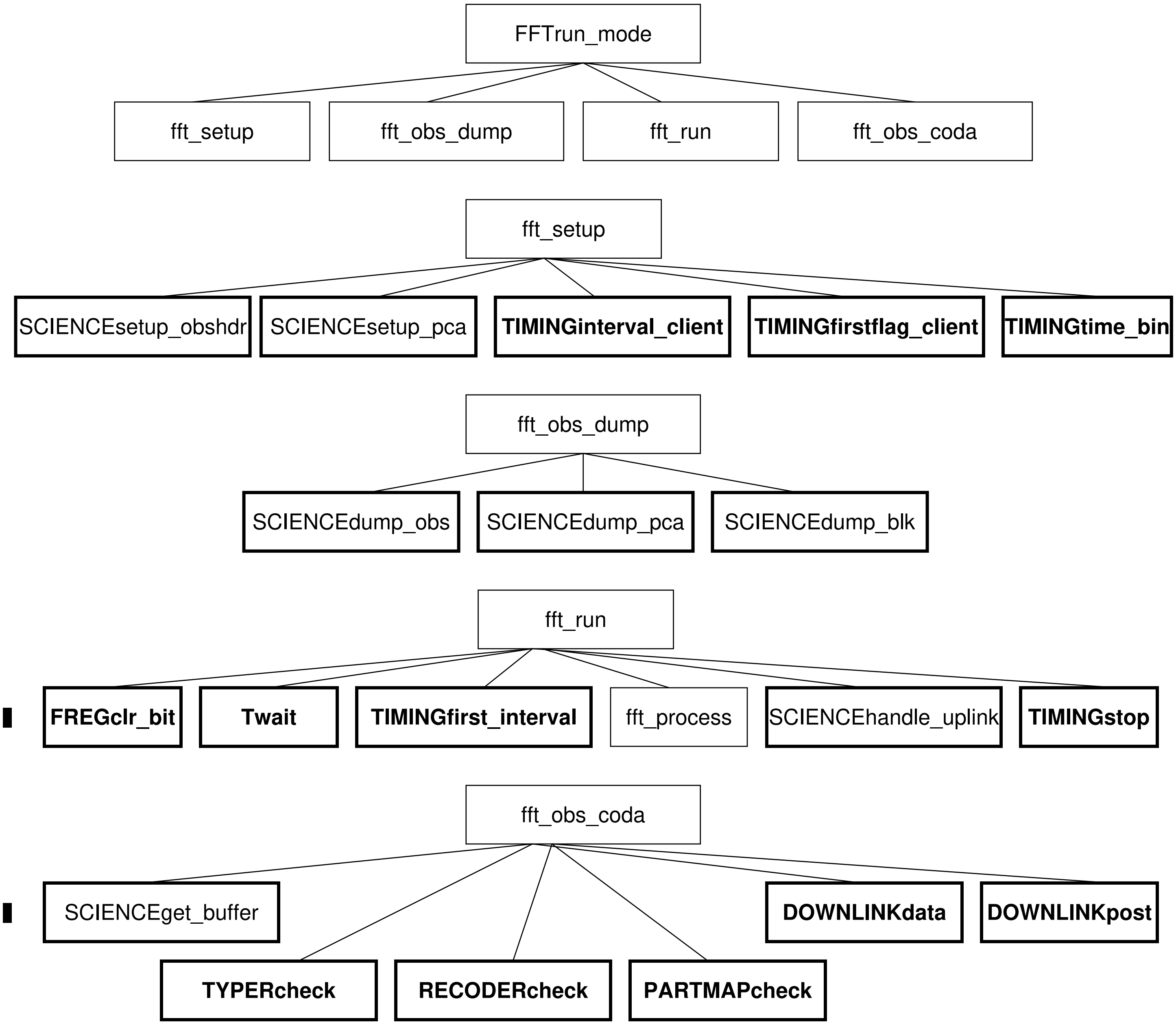


Figure 4 FFT Mode CPU Function Structure

Figure 4 does not show calls to the **Interrupt Manager** functions INTRbegin_atomic() and INTRend_atomic(), nor to the **Utility** function UTILpanic(). The following lists each of the functions shown in Figure 4. The details of each function's operation and arguments are described in FFT Mode's detailed design document.

FFTrun_mode .... Run FFT Mode Observation.
fft_setup .... Setup state variables and hardware.
fft_obs_dump .... Telemeter Parameter Set.
fft_run .... Start and manage data acquisition.
fft_obs_coda .... Check parameters and send Observation Coda.

4.5.2 Telemetry Function Structures

The following diagram illustrates the operations which occur as a result of a call to fft_process. Bolded boxes with bolded text represent functions provided by modules other than the **Science Module** or FFT Mode. Bolded boxes with unbolded text represent utility functions provided by the **Science Thread Module**. All other boxes represent functions which are private to FFT Mode.

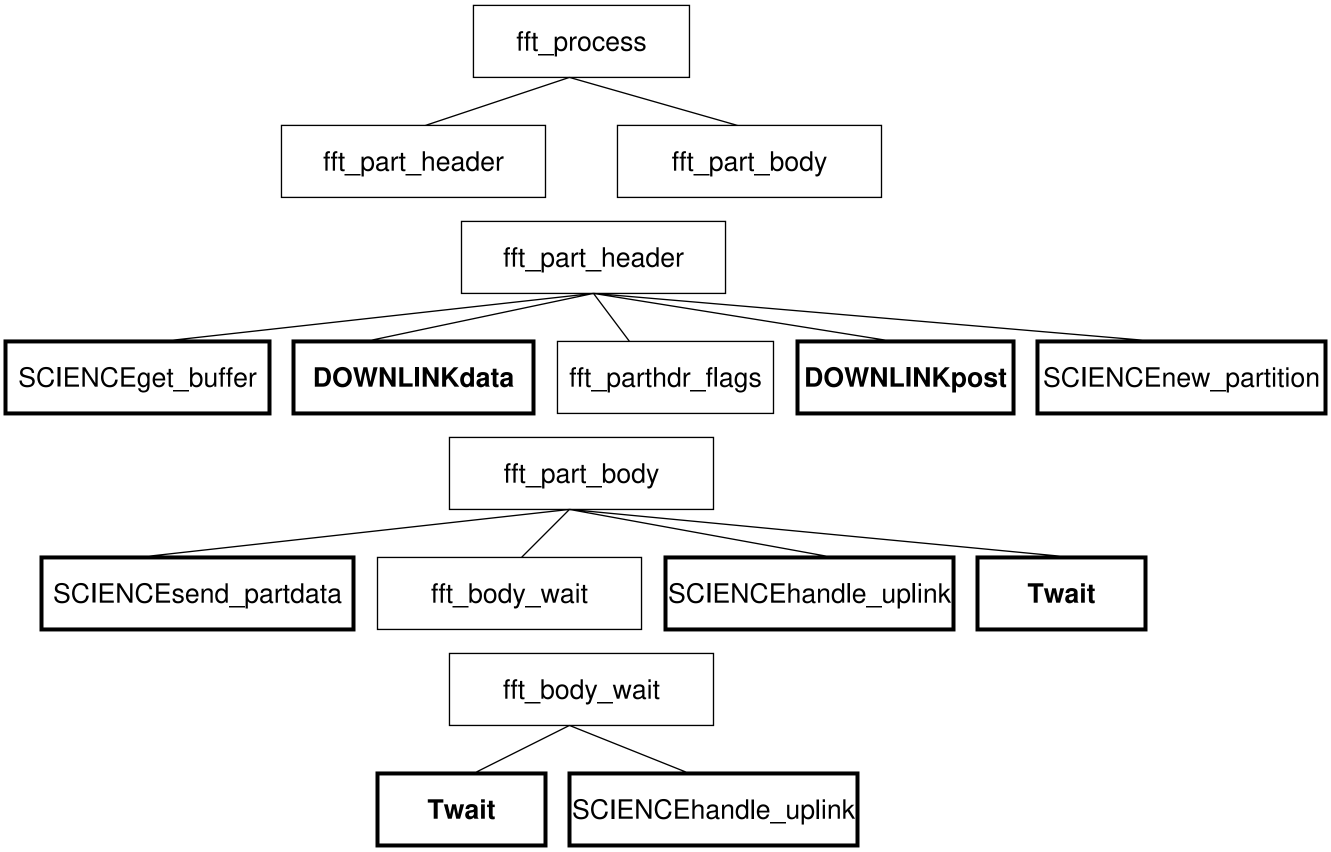


Figure 5 Data Telemetry Function Structures

Figure 5 does not show calls to the **Interrupt Manager** functions INTRbegin_atomic() and INTRend_atomic(), nor to the **Utility** function UTILpanic(). The following lists each of the functions shown in Figure 5. The details of each function's operation and arguments are described in FFT Mode's detailed design document.

fft_process .... Telemeter 1 partition of data
fft_part_header.................................. Form and telemeter a Partition Header
fft_parthdr_flags .... Form the Partition Header flags
fft_part_body .... Form and telemeter the Partition Science Data
fft_body_wait.................................... Wait for an event while handling uplinked commands

4.5.3 Interrupt Client Function Structures

This section describes the structure of the Interval Interrupt and First Flag Interrupt Client functions. Bolded boxes represent functions provided by modules other than the **Science Module** or FFT Mode.

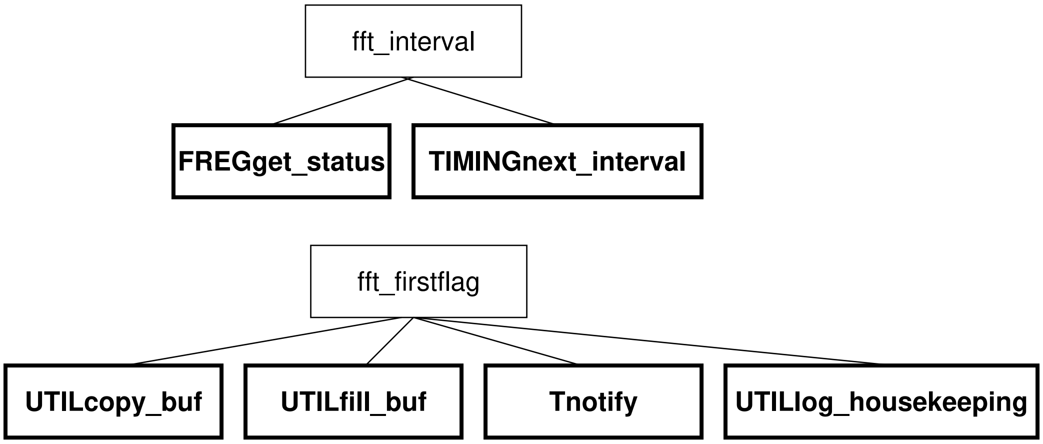


Figure 6 Interrupt Client Function Structures

Figure 6 does not show calls to the **Interrupt Manager** functions INTRbegin_atomic() and INTRend_atomic(), nor to the **Utility** function UTILpanic(). The following lists each of the functions shown in Figure 6. The details of each function's operation and arguments are described in FFT Mode's detailed design document.

fft_interval .... Schedule Intervals and maintain Interval Information
fft_firstflag.. Acquire data from DSP and reset DSP Partition

4.5.4 FFT Mode DSP Code Flow Charts

The following sections describe, in some detail, the operation of the DSP in FFT Mode. These sections describe the top level loop, and some of the subroutines used by the DSP. A detailed description of the FFT algorithm is not provided in the following descriptions.

4.5.4.1 FFT Algorithm Notes

The algorithm used by FFT Mode to compute the complex spectra of the two time-series histograms, takes real valued time-series inputs and produces complex outputs, using an integer FFT transform. Since the inputs to the algorithm are real, it is possible to compute the FFT's of both time-series histograms in one step, using the first time-series as the real input components, and second time-series as the imaginary components. The resulting real and imaginary components of the FFT are then extracted, in appropriate fashion, by the DSP, and are used in the Power spectra, and Cross spectra computations (see Sections 4.5.6 and 4.5.7). For more information on this and other techniques, see **Numerical Recipies in C**, published by Cambridge University Press.

4.5.4.2 Main Loop

The following diagram illustrates the top-level operation of the DSP when running in FFT Mode.

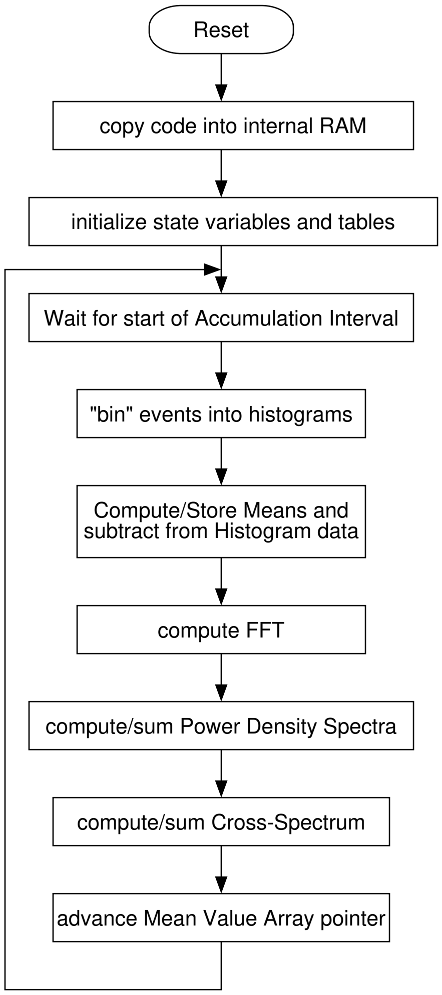


Figure 7 FFT Mode DSP Code Top-level Flow Chart

The DSP starts by copying the sections of its code and data into its internal RAM, so that the code can run as fast as possible. The DSP then enters its main loop.

At the top of the loop, the DSP branches to a routine which waits for the start of a Data Accumulation Interval. This routine is also responsible for detecting the start of a Transfer Interval and for resetting the Mean Value Array pointer whenever a Transfer Interval begins.

Once a Data Accumulation Interval starts, the DSP branches to a routine which reads events from the Input FIFO and forms two time-series histograms from the read events. This routine returns when the Processing Interval begins.

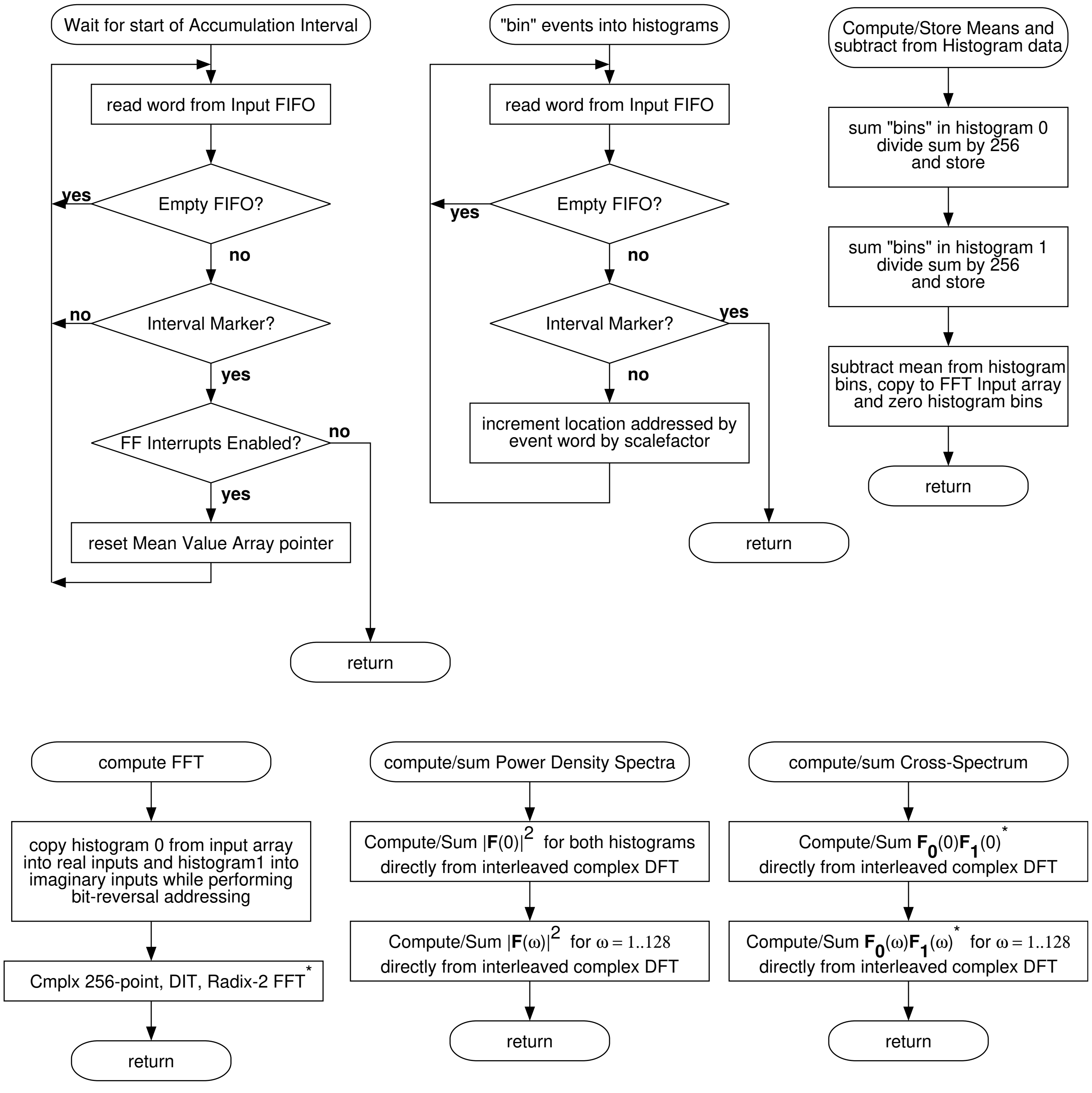
Once the Processing Interval starts, the DSP branches to a series of routines which process the time-series histograms. It starts by computing the mean values of each of the two time-series histograms, and storing the values in the current Mean Value Array. It then subtracts the computed means from their respective histograms. The DSP then computes Discrete Fourier Transforms (DFT) of the mean-subtracted histograms, using an Fast Fourier Transform (FFT) algorithm. The DSP then uses the resulting DFTs to compute and sum the Power Density Spectrum of each histogram, and the Complex Cross-Spectrum between the two histograms.

Once the processing is complete, the DSP advances the Mean Value Array pointer and branches to the top of the loop.

The DSP continues this loop until it is reset.

4.5.4.3 Subroutines

The following illustrates the operation of some of the DSP's internal routines:



*DIT, Radix-2 FFT - Decimation-in-Time algorithm (bit-reversed in time-domain), which sub-divides data sets into pairs

Figure 8 FFT DSP Code Subroutine Flow Chart

4.5.5 Complex FFT Results

FFT Mode uses a Complex, 256-point, Decimation-in-Time (DIT) Fast Fourier Transform (FFT). The inputs to the FFT, **f**(n), is an array of 256 complex numbers, where the real components are the mean-subtracted values from Histogram 0, g(n), and the imaginary components are the mean-subtracted values from Histogram 1, h(n):

$$\mathbf{f}(n) = f_{real}(n) + jf_{imag}(n) = g(n) + jh(n) \qquad \text{where } j = (-1)^{1/2} \tag{1}$$

The FFT algorithm used by this mode expects the complex input values to be "scrambled" using bit-reversal addressing. The TI-320C25 provides an addressing mode which allows the interleaving of g(n) and h(n) and "scrambling" operation to take place at the same time.

The outputs of the FFT, **F**(ω), are not scrambled; however the transform of the histogram values, **G**(ω) and **H**(ω) are "intertwined":

$$\begin{aligned}
\mathbf{F}(\omega) &= \Sigma\mathbf{f}(n)e^{-j2\pi\omega n/N} \qquad \text{for } n = 0..N\text{-}1 \\
&= \Sigma(g(n) + jh(n))(\cos(2\pi\omega n/N) - j\sin(2\pi\omega n/N)) \\
&= \Sigma(g(n)\cos(2\pi\omega n/N) + h(n)\sin(2\pi\omega n/N)) + \\
&\quad j\Sigma(h(n)\cos(2\pi\omega n/N) - g(n)\sin(2\pi\omega n/N)) && (2) \\
F_{real}(\omega) &= \Sigma g(n)\cos(2\pi\omega n/N) + \Sigma h(n)\sin(2\pi\omega n/N) && (3) \\
F_{imag}(\omega) &= \Sigma h(n)\cos(2\pi\omega n/N) - \Sigma g(n)\sin(2\pi\omega n/N) && (4)
\end{aligned}$$

Note that:

$$\begin{aligned}
\mathbf{F}(N\text{-}\omega) &= \Sigma\mathbf{f}(n)e^{-j2\pi(N-\omega)n/N} \qquad \text{for } n = 0..N\text{-}1 \\
&= \Sigma\mathbf{f}(n)e^{-j2\pi nN/N}e^{j2\pi\omega n/N} \\
&= \Sigma\mathbf{f}(n)e^{j2\pi\omega n/N} \qquad \text{(because "n" is an integer } e^{-j2\pi n} = 1 + j0) \\
&= \Sigma(g(n) + jh(n))(\cos(2\pi\omega n/N) + j\sin(2\pi\omega n/N)) \\
&= \Sigma(g(n)\cos(2\pi\omega n/N) - h(n)\sin(2\pi\omega n/N)) + \\
&\quad j\Sigma(h(n)\cos(2\pi\omega n/N) + g(n)\sin(2\pi\omega n/N)) && (5) \\
F_{real}(N\text{-}\omega) &= \Sigma g(n)\cos(2\pi\omega n/N) - \Sigma h(n)\sin(2\pi\omega n/N) && (6) \\
F_{imag}(N\text{-}\omega) &= \Sigma h(n)\cos(2\pi\omega n/N) + \Sigma g(n)\sin(2\pi\omega n/N) && (7)
\end{aligned}$$

Using the transforms of g(n) and h(n):

$$\begin{aligned}
\mathbf{G}(\omega) &= \Sigma g(n)e^{-j2\pi\omega n/N} \\
&= \Sigma g(n)\cos(2\pi\omega n/N) - j\Sigma g(n)\sin(2\pi\omega n/N) \text{ for } n = 0..N\text{-}1 \\
\mathbf{H}(\omega) &= \Sigma h(n)e^{-j2\pi\omega n/N} \\
&= \Sigma h(n)\cos(2\pi\omega n/N) - j\Sigma h(n)\sin(2\pi\omega n/N) \text{ for } n = 0..N\text{-}1 \\
G_{real}(\omega) &= \Sigma(g(n)\cos(2\pi\omega n/N) && (8) \\
G_{imag}(\omega) &= -\Sigma(g(n)\sin(2\pi\omega n/N) && (9)
\end{aligned}$$

$$H_{real}(\omega) = \Sigma(h(n)\cos(2\pi\omega n/N) \quad (10)$$
$$H_{imag}(\omega) = -\Sigma(h(n)\sin(2\pi\omega n/N) \quad (11)$$

Solving **G**(ω) and **H**(ω) in terms of **F**(ω):

$$\begin{aligned} G_{real}(\omega) &= \Sigma(g(n)\cos(2\pi\omega n/N) \\ &= 1/2(\Sigma g(n)\cos(2\pi\omega n/N) + \Sigma h(n)\sin(2\pi\omega n/N)) + \\ &\quad \Sigma g(n)\cos(2\pi\omega n/N) - \Sigma h(n)\sin(2\pi\omega n/N)) \\ &= 1/2(F_{real}(\omega) + F_{real}(N\text{-}\omega)) \qquad (12) \end{aligned}$$

$$\begin{aligned} G_{imag}(\omega) &= -\Sigma(g(n)\sin(2\pi\omega n/N) \\ &= 1/2(\Sigma h(n)\cos(2\pi\omega n/N) - \Sigma g(n)\sin(2\pi\omega n/N) - \\ &\quad \Sigma h(n)\cos(2\pi\omega n/N) - \Sigma g(n)\sin(2\pi\omega n/N)) \\ &= 1/2(F_{imag}(\omega) - F_{imag}(N\text{-}\omega)) \qquad (13) \end{aligned}$$

$$\begin{aligned} H_{real}(\omega) &= \Sigma(h(n)\cos(2\pi\omega n/N) \\ &= 1/2(\Sigma h(n)\cos(2\pi\omega n/N) + \Sigma g(n)\sin(2\pi\omega n/N) + \\ &\quad \Sigma h(n)\cos(2\pi\omega n/N) - \Sigma g(n)\sin(2\pi\omega n/N)) \\ &= 1/2(F_{imag}(N\text{-}\omega) + F_{imag}(\omega)) \qquad (14) \end{aligned}$$

$$\begin{aligned} H_{imag}(\omega) &= -\Sigma(h(n)\sin(2\pi\omega n/N) \\ &= 1/2(\Sigma g(n)\cos(2\pi\omega n/N) - \Sigma h(n)\sin(2\pi\omega n/N) - \\ &\quad \Sigma g(n)\cos(2\pi\omega n/N) - \Sigma h(n)\sin(2\pi\omega n/N)) \\ &= 1/2(F_{real}(N\text{-}\omega) - F_{real}(\omega)) \qquad (15) \end{aligned}$$

4.5.6 Power Density Spectra Computation

The Power Density Spectra are computed by taking the magnitude squared for each histogram result at each frequency and summing the result across each Product Interval. The following solves for the magnitude squared, for each input histogram, directly from the results of the "interwined" FFT, where **G**(ω) is the DFT of histogram 0, **H**(ω) is the DFT of histogram 1, and **F**(ω) is the "interwined" complex result from the FFT (see Section 4.5.5):

$$\begin{aligned} |\mathbf{G}(\omega)|^2 &= G_{real}(\omega)^2 + G_{imag}(\omega)^2 \\ &= (1/2(F_{real}(\omega) + F_{real}(N\text{-}\omega)))^2 + (1/2(F_{imag}(\omega) - F_{imag}(N\text{-}\omega)))^2 \\ &\qquad \text{(from (12) and (13))} \\ &= 1/4(F_{real}(\omega)^2 + F_{real}(N\text{-}\omega)^2 + 2F_{real}(\omega)F_{real}(N\text{-}\omega) + \\ &\quad F_{imag}(\omega)^2 + F_{imag}(N\text{-}\omega)^2 - 2F_{imag}(\omega)F_{imag}(N\text{-}\omega)) \\ &= 1/4(F_{real}(\omega)^2 + F_{real}(N\text{-}\omega)^2 + F_{imag}(\omega)^2 + F_{imag}(N\text{-}\omega)^2 + \\ &\quad 2(F_{real}(\omega)F_{real}(N\text{-}\omega) - F_{imag}(\omega)F_{imag}(N\text{-}\omega)) \qquad (16) \end{aligned}$$

$$\begin{aligned} |\mathbf{H}(\omega)|^2 &= H_{real}(\omega)^2 + H_{imag}(\omega)^2 \\ &= (1/2(F_{imag}(N\text{-}\omega) + F_{imag}(\omega)))^2 + (1/2(F_{real}(N\text{-}\omega) - F_{real}(\omega)))^2 \\ &\qquad \text{(from (14) and (15))} \end{aligned}$$

$$
\begin{aligned}
&= 1/4(F_{imag}(N-\omega)^2 + F_{imag}(\omega)^2 + 2F_{imag}(N-\omega)F_{imag}(\omega) + \\
&\quad F_{real}(N-\omega)^2 + F_{real}(\omega)^2 - 2F_{real}(N-\omega)F_{real}(\omega)) \\
&= 1/4(F_{real}(\omega)^2 + F_{real}(N-\omega)^2 + F_{imag}(\omega)^2 + F_{imag}(N-\omega)^2 + \\
&\quad 2(F_{imag}(N-\omega)F_{imag}(\omega) - F_{real}(N-\omega)F_{real}(\omega))) \qquad (17)
\end{aligned}
$$

Say:

$$X(\omega) = F_{real}(\omega)^2 + F_{real}(N-\omega)^2 + F_{imag}(\omega)^2 + F_{imag}(N-\omega)^2 \qquad (18)$$

$$Y(\omega) = F_{real}(\omega)F_{real}(N-\omega) - F_{imag}(\omega)F_{imag}(N-\omega) \qquad (19)$$

Then from (16), (17), (18) and (19):

$$|\mathbf{G}(\omega)|^2 = 1/4(X(\omega) + 2Y(\omega)) \qquad (20)$$

$$|\mathbf{H}(\omega)|^2 = 1/4(X(\omega) - 2Y(\omega)) \qquad (21)$$

The above ((20) and (21)) are solved explictly for ω = 1..N/2 (NOTE: At ω = N/2, (N-ω) = N/2. Although optimization is possible, the DSP does not optimize the computation at this point). For ω = 0, the results should be zero (the time-series histograms were mean-subtracted). For purposes of verification, the zero'th components are computed anyway. Assuming that the transform **F**(ω) is periodic with period N (**F**(N) is never explicitly computed):

$$\mathbf{F}(0) = \mathbf{F}(N) \qquad (22)$$

$$
\begin{aligned}
X(0) &= F_{real}(0)^2 + F_{real}(N-0)^2 + F_{imag}(0)^2 + F_{imag}(N-0)^2 && \text{(from 18)} \\
&= F_{real}(0)^2 + F_{real}(0)^2 + F_{imag}(0)^2 + F_{imag}(0)^2 \\
&= 2(F_{real}(0)^2 + F_{imag}(0)^2) \\
Y(0) &= F_{real}(0)F_{real}(N-0) - F_{imag}(0)F_{imag}(N-0) && \text{(from 19)} \\
&= F_{real}(0)^2 - F_{imag}(0)^2
\end{aligned}
$$

$$
\begin{aligned}
|\mathbf{G}(0)|^2 &= 1/4(X(\omega) + 2Y(\omega)) && \text{(from 20)} \\
&= 1/4(2(F_{real}(0)^2 + F_{imag}(0)^2) + 2(F_{real}(0)^2 - F_{imag}(0)^2) \\
&= F_{real}(0)^2 && (23)
\end{aligned}
$$

$$
\begin{aligned}
|\mathbf{H}(0)|^2 &= 1/4(X(\omega) - 2Y(\omega)) && \text{(from 21)} \\
&= 1/4(2(F_{real}(0)^2 + F_{imag}(0)^2) - 2(F_{real}(0)^2 - F_{imag}(0)^2) \\
&= F_{imag}(0)^2 && (24)
\end{aligned}
$$

Given that the initial time-series histogram data was scaled up, the resulting Power Density Spectra are left scaled up by the square of this factor prior to the application of the output scaling factor (see Sections 1.3 and 2.2).

4.5.7 Complex Cross-Spectrum Computation

The Complex Cross-Spectrum is computed by multiplying the complex DFT result from one histogram by the complex conjugate of the DFT of second histogram. The following solves for this product in terms of the results of the "interwined" FFT, where **G**(ω) is the DFT of histogram 0, **H**(ω) is the DFT of histogram 1, and **F**(ω) is the "interwined" complex result from the FFT (see Section 4.5.5), and **C**(ω) is the Complex Cross-Spectrum:

$$\begin{aligned}
\mathbf{C}(\omega) &= \mathbf{G}(\omega)\mathbf{H}(\omega)^* \qquad \text{where } \mathbf{H}(\omega)^* \text{ is the complex conjugate of } \mathbf{H}(\omega) \\
&= (G_{real}(\omega) + jG_{imag}(\omega))(H_{real}(\omega) + jH_{imag}(\omega))^* \\
&= (G_{real}(\omega) + jG_{imag}(\omega))(H_{real}(\omega) - jH_{imag}(\omega)) \\
&= (G_{real}(\omega)H_{real}(\omega) + G_{imag}(\omega)H_{imag}(\omega)) + \\
&\quad j(G_{imag}(\omega)H_{real}(\omega) - G_{real}(\omega)H_{imag}(\omega)) \qquad (25)
\end{aligned}$$

$$C_{real}(\omega) = G_{real}(\omega)H_{real}(\omega) + G_{imag}(\omega)H_{imag}(\omega) \qquad (26)$$
$$C_{imag}(\omega) = G_{imag}(\omega)H_{real}(\omega) - G_{real}(\omega)H_{imag}(\omega) \qquad (27)$$

Solving in terms of **F**(ω) and using some short-hand:

Set:

$$\begin{aligned}
A &= F_{real}(\omega) \\
B &= F_{real}(N-\omega) \\
C &= F_{imag}(\omega) \\
D &= F_{imag}(N-\omega)
\end{aligned}$$

$$\begin{aligned}
C_{real}(\omega) &= G_{real}(\omega)H_{real}(\omega) + G_{imag}(\omega)H_{imag}(\omega) \\
&= 1/2(F_{real}(\omega) + F_{real}(N-\omega))1/2(F_{imag}(N-\omega) + F_{imag}(\omega)) + \\
&\quad 1/2(F_{imag}(\omega) - F_{imag}(N-\omega))1/2(F_{real}(N-\omega) - F_{real}(\omega)) \\
&= 1/4(A + B)(C + D) + 1/4(C - D)(B - A) \\
&= 1/4(AC + BD + BC + AD + BC + AD - BD - AC) \\
&= 1/4(2AD + 2BC) \\
&= 1/2(F_{real}(\omega)F_{imag}(N-\omega) + F_{real}(N-\omega)F_{imag}(\omega)) \qquad (28)
\end{aligned}$$

$$\begin{aligned}
C_{imag}(\omega) &= G_{imag}(\omega)H_{real}(\omega) - G_{real}(\omega)H_{imag}(\omega) \\
&= 1/2(F_{imag}(\omega) - F_{imag}(N-\omega))1/2(F_{imag}(N-\omega) + F_{imag}(\omega)) - \\
&\quad 1/2(F_{real}(\omega) + F_{real}(N-\omega))1/2(F_{real}(N-\omega) - F_{real}(\omega)) \\
&= 1/4(C - D)(C + D) - 1/4(A + B)(B - A) \\
&= 1/4(C^2 - D^2 + A^2 - B^2) \\
&= 1/4(F_{real}(\omega)^2 + F_{imag}(\omega)^2 - F_{real}(N-\omega)^2 - F_{imag}(N-\omega)^2) \qquad (29)
\end{aligned}$$

The above ((28) and (29)) are solved explictly for ω = 1..N/2 (NOTE: At ω = N/2, (N-ω) = N/2. Although optimization is possible, the DSP does not optimize the computation at this point). For ω = 0, the results should be zero (the time-series histograms were mean-subtracted). For purposes of verification, the zero'th components are computed anyway. Assuming that the transform $\mathbf{F}(\omega)$ is periodic with period N ($\mathbf{F}(N)$ is never explicitly computed):

$$\mathbf{F}(0) = \mathbf{F}(N) \qquad \text{(from (22))}$$

$$\begin{aligned} C_{real}(0) &= 1/2(F_{real}(0)F_{imag}(N-0) + F_{real}(N-0)F_{imag}(0)) && \text{(from (28))} \\ &= 1/2(F_{real}(0)F_{imag}(0) + F_{real}(0)F_{imag}(0)) \\ &= F_{real}(0)F_{imag}(0) && (30) \end{aligned}$$

$$\begin{aligned} C_{imag}(0) &= 1/4(F_{real}(0)^2 + F_{imag}(0)^2 - F_{real}(N-0)^2 - F_{imag}(N-0)^2) && \text{(from (29))} \\ &= 1/4(F_{real}(0)^2 + F_{imag}(0)^2 - F_{real}(0)^2 - F_{imag}(0)^2) \\ &= 0 && (31) \end{aligned}$$

Given that the initial time-series histogram data was scaled up, the resulting Complex Cross-Spectrum is left scaled up by the square of this factor prior to the application of the output scaling factor (see Sections 1.3 and 2.2).

4.5.8 Accumulated Rounding Error

A 256 point FFT is performed in 8 stages (i.e. $\log_2(256)$), where data from each stage is summed with previous data. The algorithm used by FFT Mode keeps data from each stage from overflowing by dividing the data at each stage by 2. Ignoring all other forms of quantization noise, this introduces a possible error of +-0.5 at each stage. After 8 stages, this leaves a possible error of +-4 in a real or imaginary frequency component. Taking the error factor into account when computing the Power Density Spectra:

$$\mathbf{G}'(\omega) = (G_{real}(\omega) \pm \mathit{err}) + j(G_{imag}(\omega) \pm \mathit{err}) \qquad (32)$$

$$\begin{aligned} |\mathbf{G}'(\omega)|^2 &= G_{real}(\omega)^2 + \mathit{err}^2 \pm 2\mathit{err}G_{real}(\omega) + G_{imag}(\omega)^2 + \mathit{err}^2 \pm 2\mathit{err}G_{imag}(\omega) \\ &= G_{real}(\omega)^2 + G_{imag}(\omega)^2 + 2(\mathit{err}^2 \pm \mathit{err}(G_{real}(\omega) + G_{imag}(\omega))) \\ &= |\mathbf{G}(\omega)|^2 + 2(\mathit{err}^2 \pm \mathit{err}(G_{real}(\omega) + G_{imag}(\omega))) && (33) \end{aligned}$$

$$\begin{aligned} \%\mathit{Perror} &= 2(\mathit{err}^2 \pm \mathit{err}(G_{real}(\omega) + G_{imag}(\omega)))/|\mathbf{G}(\omega)|^2 \\ &= (32 \pm 4(G_{real}(\omega) + G_{imag}(\omega)))/|\mathbf{G}(\omega)|^2 && (34) \end{aligned}$$

The error for the Cross-Spectrum:

$$\mathbf{C}'(\omega) = \mathbf{G}'(\omega)\mathbf{H}'(\omega)^* \qquad \text{(from (25))}$$

$$\begin{aligned} C'_{real}(\omega) &= G'_{real}(\omega)H'_{real}(\omega) + G'_{imag}(\omega)H'_{imag}(\omega) && \text{(from (26))} \\ &= G_{real}(\omega)H_{real}(\omega) \pm \mathit{err}^2 \pm \mathit{err}(G_{real}(\omega) + H_{real}(\omega)) + \\ &\quad G_{imag}(\omega)H_{imag}(\omega) \pm \mathit{err}^2 \pm \mathit{err}(G_{imag}(\omega) + H_{imag}(\omega)) \end{aligned}$$

$$
\begin{aligned}
&= C_{real}(\omega) + \\
&\quad 2err^2 \pm err(G_{real}(\omega) + H_{real}(\omega) + G_{imag}(\omega) + H_{imag}(\omega)) && (35) \\
C'_{imag}(\omega) &= G'_{imag}(\omega)H'_{real}(\omega) - G'_{real}(\omega)H'_{imag}(\omega) && \text{(from (27))} \\
&= G_{imag}(\omega)H_{real}(\omega) \pm err^2 \pm err(G_{imag}(\omega) + H_{real}(\omega)) - \\
&\quad G_{real}(\omega)H_{imag}(\omega) \pm err^2 \pm err(G_{real}(\omega) + H_{imag}(\omega)) \\
&= C_{imag}(\omega) + \\
&\quad 2err^2 \pm err(G_{real}(\omega) + H_{real}(\omega) + G_{imag}(\omega) + H_{imag}(\omega)) && (36) \\
\mathbf{C}'(\omega) &= \mathbf{C}(\omega) + \\
&\quad 2err^2 \pm err(G_{real}(\omega) + H_{real}(\omega) + G_{imag}(\omega) + H_{imag}(\omega)) && (37) \\
\%Cerror &= 2err^2 \pm err(G_{real}(\omega)+H_{real}(\omega)+G_{imag}(\omega)+H_{imag}(\omega))/\mathbf{C}(\omega) \\
&= 32 \pm 4((G_{real}(\omega)+H_{real}(\omega)+G_{imag}(\omega)+H_{imag}(\omega))/\mathbf{C}(\omega) && (38)
\end{aligned}
$$

(NOTE: For a detailed discussion of the quantization effects in Fixed-Point FFT Algorithms, see "Digitial Signal Processing", by Oppenheim/Schafer, Prentice-Hall, 1975, Chapter 9.5.2 "Analysis of Quantization Effects in Fixed-Point FFT Algorithms).

5 Requirements Traceability

5.1 Science Requirements

FFT Mode addresses all Level 0 PCA Science Requirements dealing with the FFT Science Mode requirements. FFT Mode does not meet the Level 1 requirement of 25usec time-bins with less than 50% dead-time. The requirements which have been addressed by this document are listed in the **Requirements Traceablity Matrix** and in the FFT Mode section of the **EDS-PCA Science Requirements**.

The ability of FFT Mode (or any mode for that matter) to govern its telemetry rate is a derived requirement. Given that it is possible that all 8 EA's may be trying 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, it is the responsibility of the ground to configure the slower EA's to 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 SDS in time to maintain their desired data rate.

6 Testing Considerations

6.1 Unit Testing

To perform unit tests on this module, the following equipment is required:

a. An Event Analyzer equipped with the Intermetrics ROM Monitor Debug Board

b. A workstation running the Intermetrics debugger

c. An RS-232 serial cable between the ROM Monitor's serial port and the workstation

d. A logic analyzer

To unit test this module, the tester must link and locate this module's functions to some EA RAM memory location. Using a resident monitor/debugger, the tester should first verify the operation of all functions internal to this module, and then verify the interactions between all functions internal to the module. These tests should verify all branches, calls, and legal and illegal parameters. The tester should then verify all calls made to functions outside of this module, and ensure that the passed arguments are always within the limits specified by the corresponding module specifications. The tester should also verify that the caller always handles values returned by these functions in an appropriate manner.

In addition to these tests, the tester should perform the following:

a. Verify that all functions which interpret components of the Observation Parameter Block and the FFT Mode Parameter Block extract the proper items and test the items for valid ranges.
These functions include fft_setup.

b. Measure the execution times of each function, given all control paths through the function. This information, along with mode specific measurements, are used to calculate maximum execution times, which are then used to ensure that the thread's temporal requirements can be met. Pay special attention to the time it takes for fft_part_data to pack and post a data set.

c. Measure the histogram processing time of the DSP's Mean Value and spectra computations. Ensure that this time is sufficiently less than 12ms (50us/bin @

256 bins = 12.8ms), such that, when combined with the transfer time (see [e]), there is less than 50% dead-time during a given observation.

d. Measure the performance of the Interval Interrupt Client function. This time contributes to the minimum Accumulation Interval, and may affect the minimum Processing Interval durations when running FFT Mode. For Accumulation Intervals, this time can affect the time-resolution of the acquired data (and possibly the dead-time introduced if events with a time-stamp greater than 255 are recorded). For Processing Intervals, this time becomes a limiting factor if the DSP can complete its processing faster than the CPU can turn around Interval Interrupts. Ensure that this function takes no longer than 1ms, under all conditions and execution paths.

e. Mesaure the performance of the First Flag Interrupt Client function. This time governs the minimum Transfer Interval Duration. Ensure that this function can transfer data fast enough, such that the sum of the processing times plus the transfer time is less than the sum of the accumulation times (i.e. there is less than 50% dead-time). Note that the total amount of data that needs to be transferred during each First-Flag Interrupt is dependent on the number of Accumulation/Processing Intervals used to produce a given data set.

f. Verify that Science Commands are received properly and result in the appropriate actions.

g. Verify that the FFT Power Density Spectrum and Complex Cross-Spectrum are correctly computed.

6.2 Integration Testing

For integration tests, the following equipment is required:

a. An Event Analyzer

b. A populated System Manager board

c. A logic analyzer

d. Hardware which can:

    1. Drive the MinorTick and MajorTick clock signals (S/C Simulator)

    2. Receive telemetry packets from the EA (S/C Simulator).

    3. Block telemetry packets from the EA (i.e. stop receiving packets).

    4. Send simulated PCA events to the EA (PCA-Simulator).

The integration tests should verify the operation of the FFT Mode in conjunction with the other FS modules. These tests should include:

a. Verify that the FFT Mode starts properly after a reboot.

b. Verify that command receipt and execution operates correctly during data acquisition and processing.

c. Measure DSP data accumulation performance, to determine the absolute maximum raw event rate supported during Accumulation Intervals.

d. Measure DSP processing performance, to determine the absolute minimum Processing Interval duration.

e. Measure Data Packing and transfer performance to determine the maximum data set transfer rate that can be supported.

f. Fabricate the overthroughput conditions, specified in Section 4.4, and verify that each condition is handled properly.

g. During FFT operations, measure Interval Interrupt handling performance, including system overhead. Ensure that the time taken to process an Interval Interrupt is less than 1ms (i.e. consumes no more than 20% of the CPU while handling 5ms Intervals).

h. Test FFT Mode using 50us time-bins and 12.8ms Interval Durations and a 10.5ms Processing Interval duration. Using 1 Accumulation/Processing Interval per data set, pick a Transfer Interval duration which ensures less than 50% dead-time and ensure that data is accumulated, processed and transferred correctly. Repeat using 7676 and 3837 Accumulation/Processing Intervals per data set.

i. Examine the FFT Mode section of the **EDS-PCA Science Requirements** and fabricate tests, using a variety of data which produces known spectra, to verify that all FFT Mode Science Requirements have been met.

Appendix A

A.1 Items Under Review

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

Document Reference	Section	Resolution Expected by
NONE		