# Pulsar Fold Mode Module Specification

James Francis

MIT 64-30112.060102

Rev. C

October 22, 1993

# Contents


1 Introduction ..... 4
1.1 Overview ..... 4
1.2 Theory of Operation ..... 4
1.3 Data Flow ..... 6
1.4 DSP ..... 6
1.5 CPU ..... 6
1.6 Accumulated Phase Error ..... 8

2 Data Structures ..... 9
2.1 Pulsar Fold Mode Parameter Block ..... 9
2.2 DSP Pulsar Fold Mode Parameter Block ..... 9
2.3 Constants ..... 9
2.3.1 PULSAR_FRAC_MICROTICK (0x010000L) ..... 9
2.3.2 PULSAR_FRAC_HALF_MICROTICK (0x08000L) ..... 9
2.3.3 PULSAR_IC_P0 (0) ..... 9
2.3.4 PULSAR_IC_P1 (2) ..... 10
2.3.5 PULSAR_OUTSTANDING_FF (2) ..... 10
2.3.6 PULSAR_PARTNUM_ADJUST (1) ..... 10
2.3.7 LOG_SCIMODE_FF_DROP_PART (SCIENCE) ..... 10
2.3.8 MSG_SCIMODE_RUN_MODE_NULL_HDR (SCIENCE) ..... 10
2.3.9 MSG_SCIMODE_PROCESS_UNLOCKED_BUF (SCIENCE) ..... 10
2.3.10 MSG_SCIMODE_PROCESS_BAD_PARTNUM (SCIENCE) ..... 10
2.3.11 MSG_SCIMODE_IV_DEAD_DSP (SCIENCE) ..... 11

3 Functions ..... 12
3.1 PULSARrun_mode() - Run a Pulsar Fold Observation ..... 12

4 Detailed Design Considerations ..... 13
4.1 Data Flow ..... 13
4.2 Control Considerations ..... 15
4.2.1 Execution States ..... 15
4.2.1.1 Hardware Event Processing States ..... 15
4.2.1.2 DSP Event Processing States ..... 15
4.2.1.3 Interval Interrupt Processing States ..... 16
4.2.1.4 First Flag Interrupt Processing States ..... 16
4.2.1.5 Telemetry Processing States ..... 17
4.2.2 Command Handling ..... 18
4.3 Interval Timing Description ..... 18
4.4 Overthroughput Descriptions ..... 21
4.4.1 Event Rate exceeds DSP Rate ..... 21

4.4.2 DSP Rate exceeds Partition Rate ..... 21
4.4.3 Partition Rate exceeds First Flag Copy Rate ..... 21
4.4.4 First Flag Copy Rate exceeds Data Packing Rate ..... 22
4.4.5 Data Packing Rate exceeds Telemetry Rate ..... 22
4.4.6 Dropped Partition Implementation Notes ..... 22
4.5 Module Structure ..... 23
4.5.1 Pulsar Fold Mode CPU Function Structure ..... 23
4.5.2 Data Telemetry Function Structures ..... 25
4.5.3 Interrupt Client Function Structures ..... 26
4.5.4 Phase Interval Duration Computation ..... 26
4.5.5 Pulsar Fold Mode DSP Code Flow Chart ..... 28

5 Requirements Traceability ..... 29
5.1 Science Requirements ..... 29

6 Testing Considerations ..... 30
6.1 Unit Testing ..... 30
6.2 Integration Testing ..... 31

Appendix A ..... 33
A.1 Items Under Review ..... 33

## Figures

Figure 1 Pulsar Fold Mode Data Flow ..... 6
Figure 2 Pulsar Fold Science Mode Data Flow ..... 13
Figure 3 Pulsar Fold Mode CPU Function Structure ..... 23
Figure 4 Data Telemetry Function Structures ..... 25
Figure 5 Interrupt Client Function Structures ..... 26
Figure 6 Pulsar Fold Mode DSP Code Flow Chart ..... 28

## Tables

Table 1 Pulsar Fold Mode Process States ..... 19

# 1 Introduction

## 1.1 Overview

This document describes the interface and top level design of the Pulsar Fold Science Mode. This document relies heavily on the **Science Thread Module Specification**, MIT 64-30112.0601 Rev. D, 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 Pulsar Fold 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 Pulsar Fold 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 Pulsar Fold Mode. Appendix A lists all items which need to be determined or reviewed.

## 1.2 Theory of Operation

The Pulsar Fold Science Mode, which only runs on PCA-EA's, produces a series of histograms of phase-stamped, typed, recoded event words, where a "*phase-stamp*" represents some measurement of time relative to a psuedo-periodic signal. For the purposes of this document, phase-stamped, typed, recoded event words are known as "*processed event words*". Each histogram is accumulated over a period of time, known as a Data Accumulation Interval, which, in turn, is broken into periods of time known as Phase Intervals, which define the psuedo-periodic signal. At the start of each Phase Interval, the Front End hardware writes an Interval Marker into the DSP's Input FIFO, 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.

First Flag ..... Indicates whether or not to cause a First-Flag Interrupt on the CPU when the DSP reads an Interval Marker from its Input FIFO.

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 Pulsar Fold Mode, only the top bit of the Interval Constant is changed. The lower bit is held to zero. The top bit is used to select a section of the DSP's histogram memory to use during a given Data Accumulation Interval.

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

During a given Phase 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 (this serves as a phase-bin value) and Interval Constant to form the processed event word. It then writes the formed word into the DSP's Input FIFO. As this occurs, the DSP reads these words from its Input FIFO, and uses the read word as an address. The contents of the memory location addressed by the processed event word is then incremented. At the end of each Phase Interval, the Front End hardware resets the time-bin counter. During the subsequent Phase Intervals of the Data Accumulation Interval, the DSP continues to read processed event words, incrementing the histogram locations addressed by the events. Since the time-bin counter is reset at the end of each Phase Interval, it serves as a phase-bin identifier. This process results in a histogram whose elements contain the number of identical processed event words taken during the Data Accumulation Interval. Since the processed event words contain a time-stamp value which is relative to the beginning of each Phase Interval, a set of phase-series histograms are formed during the Data Accumulation Interval. (NOTE: a Data Accumulation Interval consisting of one Phase Interval results in a time-series histogram, similar to that described in the **Binned Mode Module Specification**).

During this process, the CPU schedules the next Phase Interval, specifying the Interval Constant to use during the scheduled Phase Interval and the duration of the scheduled Interval. The CPU switches the Interval Constant at each Data Accumulation Interval boundary, such that the DSP accumulates histograms in alternating buffers for each Data Accumulation Interval. As the DSP reads an Interval Marker from its Input FIFO at the start of a Data Accumulation Interval, a First-Flag Interrupt is generated on the CPU. The CPU uses this interrupt to indicate that the DSP has finished processing all processed events which were detected during the previous Data Accumulation Interval. The CPU then copies the completed histogram from the DSP's memory to the CPU's Science Buffer. The CPU then zeros the contents of the DSP's histogram. During the rest of the running Interval, the CPU packs and telemeters the data copied from the previous Data Accumulation Interval.

## 1.3 Data Flow

The following diagram illustrates the Pulsar Fold Mode data flow:

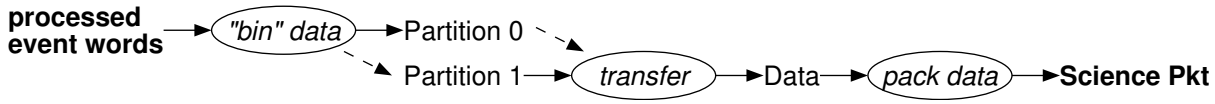


Figure 1 Pulsar Fold Mode Data Flow

## 1.4 DSP

Initially, all of the DSP's histogram memory is set to zero. Once running, the operation of the DSP Pulsar Fold Mode code consists of a single loop. The DSP merely reads "*processed event words*" from the DSP's Input FIFO, and increments the histogram memory location addressed by the event word. Once started, the DSP continues this operation until it is reset. The histogram's memory is partitioned by the event word's Interval Constant, which is switched by the CPU before the start of each Data Accumulation Interval. If the Input FIFO is empty, the hardware generates a fixed event code which maps to a specific memory location in the DSP. This allows the DSP to run continuously, without ever needing to branch based on the data read from the Input FIFO.

The DSP Code needed to support Pulsar Fold Mode is identical to that used by the **Binned Science Mode** (see **Binned Mode Module Specification**, Section 1.4).

## 1.5 CPU

As in every Science Mode, Pulsar Fold Mode consist of four stages; setup, dump, run, and check. In the setup stage, the CPU side of Pulsar Fold Mode uses the Science Thread utility functions to initialize the DSP, Partition Map, Typer Table, and the Recoder Table. The Science Thread utility function which loads the DSP code also zeros the entire DSP histogram buffer. In addition to these items, Pulsar Fold Mode uses the **Timing Module** to install Pulsar Fold 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, Pulsar Fold Mode uses the Science Thread utility functions to dump the contents of the Observation Parameter Block Header and PCA Mode Specific Data, and all components referenced by these structures, to the ground.

During the run stage, Pulsar Fold Mode schedules the first Phase Interval of the observation, enables event acquisition, and sets the Interval Constant such

that data is accumulated in DSP Partition 0. Pulsar Fold Mode then enters its main processing loop. During this loop, Pulsar Fold Mode waits for the First-Flag Interrupt client function (see below for a description on when this is enabled) to notify the thread that a partition of 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), Pulsar Fold Mode packs the copied partition of data into telemetry packets and posts the packets for transfer, using the Science Thread utility function, SCIENCEsend_partdata. If Pulsar Fold 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 Pulsar Fold Mode is not configured to govern its telemetry usage, Pulsar Fold Mode waits only when there are no telemetry packets available to send the data. It does, however, yield control after 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, Pulsar Fold Mode flags the Science Buffer as ready to take another partition, and waits for a new partition to arrive. If science telemetry is suspended, Pulsar Fold Mode sends only the partition header packets (see **Science Thread Module Specification**, "Partition Header Packet") for each partition of data acquired.

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

When the first Phase Interval starts, the hardware generates an Interval Interrupt, and the Pulsar Fold Mode Interval Interrupt client function is invoked. The Interval Interrupt client function then schedules the second Phase Interval, adjusting the duration of the interval to minimize the accumulated error relative to the specified period duration. When the first Phase Interval ends and the second starts, another Interval Interrupt occurs, and the Interval client function schedules the third Phase Interval of the Data Accumulation Interval.

When the Interval Interrupt client function is invoked by the start of the last Phase Interval of the Data Accumulation Interval, the client function then

schedules the first Phase Interval of the next Data Accumulation Interval by switching the Interval Constant, which causes the DSP to accumulate data in Partition 1, and by enabling First Flag interrupt generation.

When the DSP reads the Interval Marker, written at the start of the second Data Accumulation Interval, the hardware generates a First-Flag Interrupt, which invokes the Pulsar Fold Mode First-Flag Interrupt client function. This client function copies the data accumulated in Partition 0 from the DSP's RAM into the Science Buffer, flags the buffer as full, notifies the Science Thread that a partition of data is ready, and zeros the contents of the DSP's Partition 0.

At the start of the third Data Accumulation Interval, when the DSP reads the Interval Marker, written at the start of of the Interval, the First-Flag client function is invoked. This time, the client function tests the state of the Science Buffer. If the buffer is "empty", the First-Flag client function copies the data accumulated during the second interval (Partition 1), flags the buffer as full, notifies the Science Thread, and zeros the contents of Partition 1. If the Science Buffer is still in use (i.e. Pulsar Fold Mode hasn't finished telemetering the contents), the First-Flag client function records the occurrence in the Software Housekeeping database, and zeros Partition 1 without copying the data.

## 1.6 Accumulated Phase Error

The objective of Pulsar Fold Mode is to count events which are detected relative to a fixed point in the cycle of a periodic X-ray source. Since the period of the source may not be an integer number of microticks, a technique is employed to minimize the accumulated phase error relative to the source. This technique uses two Interval durations, which are one microtick apart. The CPU schedules the two Interval durations such that the mean approximates the period of the source for the duration of the observation. For example, if the desired period is 10,000.7 microticks, the two interval durations would be 10,000 and 10,001 microticks. Each Interval is scheduled with one of these two values such that the absolute accumulated error is always at a minimum.

This scheme has the side-effect of effectively varying the duration of the last phase-bin by one microtick for some Phase Intervals of the Data Accumulation Interval. The effect of this is not discussed in this document.

# 2 Data Structures

## 2.1 Pulsar Fold Mode Parameter Block

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

Integer Interval Duration (4 bytes) .. Specifies an integer number of microticks in a Phase Interval. The ground must ensure that this value is within operational limits (see Section 6.2).

Fract. Interval Duration (2 bytes) .... Specifies the fractional portion of a Phase Interval. The value represents the number of $2^{-16}$ microticks.

Folds (2 bytes) ..... Specifies the number of Phase Intervals per Data Accumulation Interval minus 1 (a value of 0 corresponds to 1 fold).

Partition Length (2 bytes) ..... Specifies the number of words in a partition of data sent by the DSP every Data Accumulation Interval.

## 2.2 DSP Pulsar Fold Mode Parameter Block

In Pulsar Fold Mode, the DSP takes no parameters.

## 2.3 Constants

### 2.3.1 PULSAR_FRAC_MICROTICK (0x010000L)

This constant defines 1 microtick in units of $2^{-16}$ microticks.

### 2.3.2 PULSAR_FRAC_HALF_MICROTICK (0x08000L)

This constant defines 1/2 a microtick in units of $2^{-16}$ microticks.

### 2.3.3 PULSAR_IC_P0 (0)

This constant defines the interval constant value used by Pulsar Fold Mode to select Partition 0 of the DSP Memory.

### 2.3.4 PULSAR_IC_P1 (2)

This constant defines the interval constant value used by Pulsar Fold Mode to select Partition 1 of the DSP Memory.

### 2.3.5 PULSAR_OUTSTANDING_FF (2)

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

### 2.3.6 PULSAR_PARTNUM_ADJUST (1)

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

### 2.3.7 LOG_SCIMODE_FF_DROP_PART (SCIENCE)

The First Flag client function, pulsar_firstflag(), increments the Software Housekeeping statistic whenever it cannot copy a partition of data because the requesite 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.

### 2.3.8 MSG_SCIMODE_RUN_MODE_NULL_HDR (SCIENCE)

PULSARrun_mode() 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.

### 2.3.9 MSG_SCIMODE_PROCESS_UNLOCKED_BUF (SCIENCE)

pulsar_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.

### 2.3.10 MSG_SCIMODE_PROCESS_BAD_PARTNUM (SCIENCE)

pulsar_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.

### 2.3.11 MSG_SCIMODE_IV_DEAD_DSP (SCIENCE)

The Interval Interrupt client function, pulsar_interval() emits this panic message whenever it suspects that the DSP is hung. Pulsar Fold 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.

# 3 Functions

## 3.1 PULSARrun_mode() - Run a Pulsar Fold Observation

**Usage**

void PULSARrun_mode (UINT8 **hdr*);

**Description**

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

This function does not return until the observation completes, or is terminated or aborted.

**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 Pulsar Fold Science Mode:

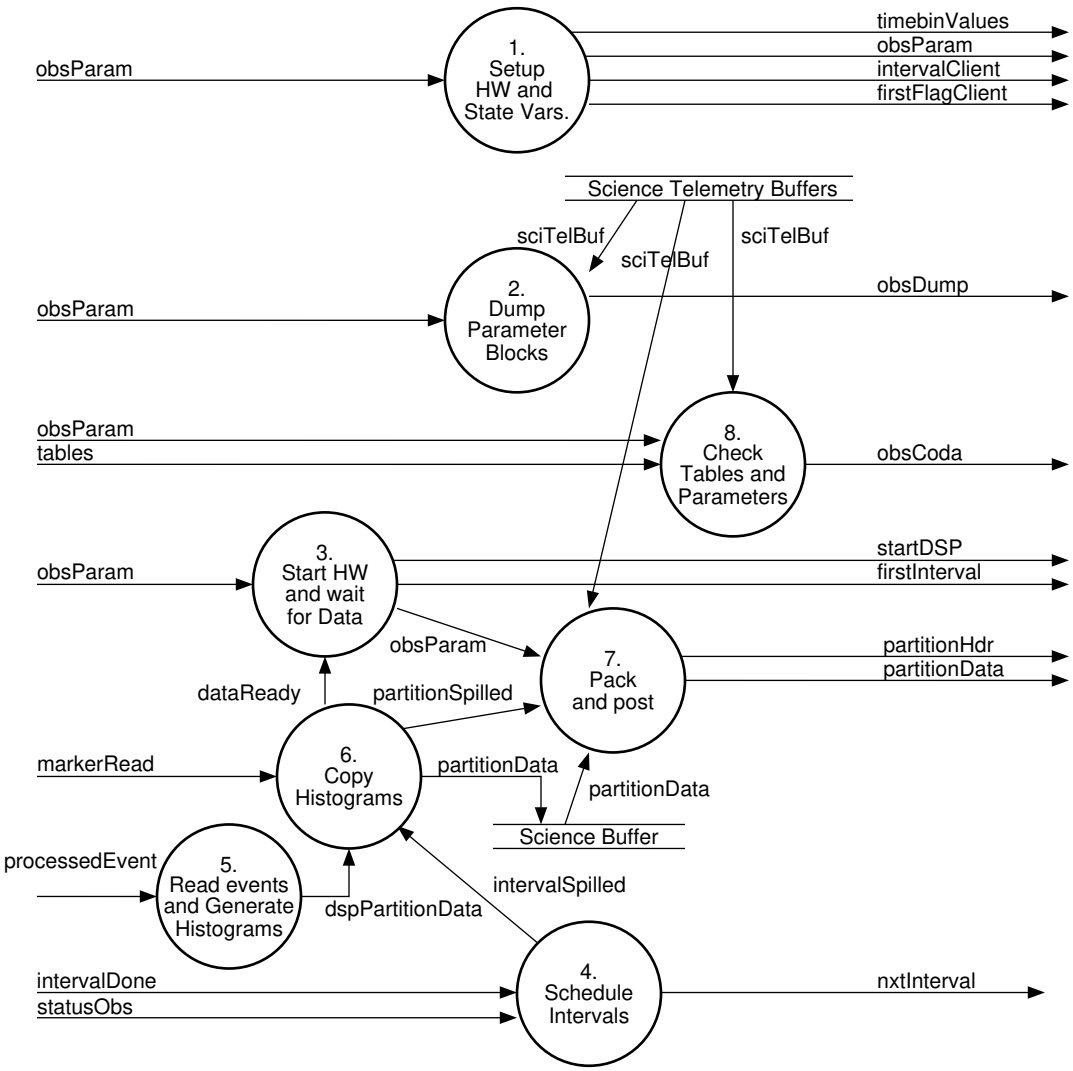


Figure 2 Pulsar Fold Science Mode Data Flow

To keep the diagram as simple as possible, Figure 2 does not show any command handling, error checking or panic messages.

The first operation performed by Pulsar Fold Mode (bubble 1) is to setup the system to perform an observation. It configures Front End hardware to run the observation, and installs the Pulsar Fold 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). Pulsar Fold 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, Pulsar Fold Mode sets up the Front End's timebin duration and mask (timebinValues), using the **Timing Module**.

Once the hardware has been configured, and the client functions have been installed, Pulsar Fold 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, Pulsar Fold Mode is ready to run the observation (bubble 3). It starts by releasing the DSP's reset line (startDSP) and scheduling the first Phase Interval of the observation (firstInterval). It then waits for completed histograms. As the timing circuitry generates interval interrupts at the start of each Phase Interval (intervalDone), the installed Interval Interrupt client function is called (bubble 4). This function reads the Observation Status Register (statusObs) to determine if the DSP's Input FIFO spilled during the completed Interval (intervalSpilled), computes the duration of the next Phase Interval of the Data Accumulation Interval and schedules the Phase Interval (nxtInterval), which starts upon the completion of the current interval. When scheduling the first Phase Interval of a Data Accumulation Interval, the Interval Interrupt Client Function enables First Flag Interrupt generation. When scheduling subsequent Phase Intervals of a Data Accumulation Interval, the client function disables First Flag Interrupt generation.

Meanwhile, the DSP reads incoming processed event words (processedEvent), and generates histograms of the read events (bubble 5). At the start of each Phase Interval, the hardware writes an Interval Marker into the DSP's Input FIFO. When the DSP reads the marker (markerRead) and First Flag Interrupt generation has been enabled for the interval, 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 next Data Accumulation Interval has started, and that the data accumulated during the previous Data Accumulation Interval is complete. The First Flag Interrupt Client function copies completed histograms (dspPartitionData) from the DSP's external RAM to the CPU's Science Buffer (partitionData). This function then zeros the DSP's partition (not shown), and notifies Pulsar Fold 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 funcction during the Data Accumulation Interval (intervalSpilled), to a flag associated with the copied data (partitionSpilled). Pulsar Fold Mode then packs the data into telemetry buffers (sciTelBuf), and posts the packets (partitionHdr, partitionData).

Once the observation is terminated, Pulsar Fold 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 Pulsar Fold Mode runs an observation, five "concurrent" processes are involved; hardware event processing, DSP event 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 P/A* ..... Start Phase Interval P of Data Accumulation Interval A.
Start an Interval, reset the timebin counter, update the Interval Constant and Spillage Flag, and write an Interval Marker into the DSP's Input FIFO. If the Interval is flagged as the first, synchronize the start of the Interval to the next MajorTick boundary. If not, start the Interval immediately as the previous Interval completes.

*Process P/A* ..... Process Phase Interval P of Accumulation Interval A.
Process events, using the current Interval Constant, 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 P/A* ..... Process Interval Marker
Read Interval Marker, written by the hardware at the start

of Phase Interval P of Data Accumulation Interval A, from the Input FIFO, causing a First Flag Interrupt on the CPU if enabled.

*Process P/A* ..... Process Events
Read events, written by the hardware during Phase Interval P of Data Accumulation A, from the Input FIFO, and generate the histogram.

#### 4.2.1.3 Interval Interrupt Processing States

*Wait P/A* ..... Wait for Interval Interrupt
Wait for the Interval Interrupt which indicates that Phase Interval P from Data Accumulation Interval A has started.

*Schedule P/A* ..... Schedule Phase Interval P of Accumulation Interval A
When Phase Interval P starts, it causes the Interval Interrupt. The Interval Interrupt Client function then schedules the next Phase Interval of the Data Accumulation Interval, or schedules the first Phase Interval of the next Data Accumulation Interval. All Phase Intervals of a given Data Accumulation Interval use the same Interval Constant. By convention, histograms accumulated during even numbered Data Accumulation Intervals use DSP Partition 0, and histograms accumulated during odd numbered Data Accumulation Intervals use DSP Partition 1 (partitions are selected using the Interval Constant).

*Terminate P/A* ..... Terminate at next Phase Interval.
When Phase Interval P starts, it causes the Interval Interrupt. The Interval Interrupt Client function then configures the system to terminate the observation after the Phase Interval completes. The histogram accumulated during the current Data Accumulation Interval is not completed.

#### 4.2.1.4 First Flag Interrupt Processing States

*Wait A* ..... Wait for First Flag Interrupt A
Wait for the First Flag Interrupt caused by the DSP reading the Interval Marker that was written at the start of the first Phase Interval of Data Accumulation Interval A. This indicates that the DSP has started processing events which occurred during Data Accumulatin Interval A, and that all events produced during the previous Data Accumulation Interval have been processed.

*Process A* ..... Process histogram produced by Accumulation Interval A.
When the DSP reads the Interval Marker written at the start of the first Phase Interval of Data Accumulation Interval A, the hardware generates a First Flag Interrupt.

The First Flag Interrupt Client function then copies and zeros the histogram built from events produced during the previous Data Accumulation Interval. By convention, histograms accumulated during even numbered Data Accumulation Intervals use DSP Partition 0, and histograms accumulated during odd numbered Data Accumulation Intervals use DSP Partition 1 (partitions are selected by the Interval Interrupt Client function using the Interval Constant).

### 4.2.1.5 Telemetry Processing States

*Start DSP* ..... Start DSP Execution
When starting the observation, Pulsar Fold 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 Pulsar Fold Mode is required.

*Schedule Phase Interval 0/0* ..... Schedule first Phase Interval.
To start the observation, Pulsar Fold Mode schedules the first Phase Interval of the first Data Accumulation Interval, using Partition 0. The Interval will start on the next MajorTick boundary which is an even multiple of 16 of the MET value.

*Wait A* ..... Wait for Histogram A
Once started, Pulsar Fold Mode waits for a histogram, produced by events which occurred during Data Accumulation Interval A, to be completed and copied from the DSP.

*Process A* ..... Process Histogram A
Once the histogram, produced by events which occurred during Data Accumulation Interval A, is ready, Pulsar Fold Mode packs and telemeters the histogram.

*Terminate* ..... Terminate Observation
When commanded to terminate an observation, Pulsar Fold Mode finishes telemetering the current histogram, and then terminates the observation.

### 4.2.2 Command Handling

Pulsar Fold 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 Pulsar Fold Mode. 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. The following table illustrates the processing steps during Pulsar Fold Mode:

| Time | Thread | HW | DSP | Interval Int. | First Flag Int. |
|---|---|---|---|---|---|
| t0 | | Disabled | Reset | Wait 0/0 | Wait 0 |
| t1 | Start DSP | | | | |
| t2 | | | Initialize | | |
| t3 | Sched 0/0 | | | | |
| t4 | Wait 0 | Start 0/0 | | | |
| t5 | | Process 0/0 | Start 0/0 | Schedule 1/0 | |
| t6 | | | Process 0/0 | | |
| t7 | | Start 1/0 | | | |
| t8 | | Process 1/0 | Start 1/0 | Schedule 2/0 | |
| t9 | | | Process 1/0 | | |
| t10 | | Start 2/0 | | | |
| t11 | | Process 2/0 | Start 2/0 | Schedule 0/1 | |
| t12 | | | Process 2/0 | | |
| t13 | | Start 0/1 | | | |
| t14 | | Process 0/1 | Start 0/1 | Schedule 1/1 | |
| t15 | | | Process 0/1 | | Process 0 |
| t16 | Process 0 | | | | |
| t17 | Wait 1 | Start 1/1 | | | |
| t18 | | Process 1/1 | Start 1/1 | Schedule 2/1 | |
| t19 | | | Process 1/1 | | |
| t20 | | Start 2/1 | | | |
| t21 | | Process 2/1 | Start 2/1 | Schedule 0/2 | |
| t22 | | | Process 2/1 | | |
| t23 | | Start 0/2 | | | |
| t24 | | Process 0/2 | Start 0/2 | Schedule 1/2 | |
| t25 | | | Process 0/2 | | Process 1 |
| t26 | Process 1 | | | | |
| t27 | Wait 2 | Start 1/2 | | | |
| t28 | Terminate | Process 1/2 | Start 1/2 | Schedule 2/2 | |
| t29 | | Disabled | Reset | | |

Table 1 Pulsar Fold Mode Process States

For the purposes of this illustration, assume that there are three Phase Intervals for each Data Accumulation Interval. Pulsar Fold 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 Phase Interval to occur, and the First Flag Interrupt Client function installed and waiting for the first Data Accumulation Interval to complete (t0). The thread starts the DSP by releasing its reset line (t1), and then waits for 1 MajorTick (via TIMINGfirst_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 the first Phase Interval of the first Data Accumulation Interval (t3).

Once the first Phase 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. Events acquired during all Phase Intervals of Data Accumulation Interval 0 are accumulated into DSP Partition 0. The Interval Interrupt Client function then computes the duration of the next Phase Interval and schedules the Phase Interval 1/0 (t5). The Interval Client function must always finish before the current Phase Interval ends (t7). At some point before the running Phase Interval ends (t7), the DSP reads the Interval Marker (t5) and processes the events which occur during Phase Interval 0/0, building a histogram into DSP Partition 0 (t6). Although the DSP must always read the marker before the Phase Interval ends, it may continue to process the Interval's events during part of the next Phase Interval.

When Phase Interval 1/0 starts (and Phase Interval 0/0 ends) (t7), the hardware writes another Interval Marker into the DSP's Input FIFO, and generates an Interval Interrupt. Events acquired during Phase Interval 1/0 are still accumulated into DSP Partition 0, "folding" the additional counts to the histogram produced during Phase Interval 0/0 (t9). The Interval Interrupt Client function computes the duration of the next Phase Interval and schedules Phase Interval 2/0 (t8), which is then processed in an identical fashion by the hardware and DSP (t10-t13). This continues until the last Phase Interval of the Data Accumulation Interval starts (t10).

When the last Phase Interval of the Data Accumulation Interval starts (t10), an Interval Interrupt is generated. The Interval Interrupt Client function then schedules the first Phase Interval of the next Data Accumulation Interval (t11). The client function sets the Interval Constant such that data is accumulated in DSP Partition 1, and enables First Flag Interrupt generation.

When the DSP reads the Interval Marker written at the start of the first Phase Interval of the second Data Accumulation Interval (t14), the hardware generates a First Flag Interrupt on the CPU. The First Flag Interrupt Client function assumes that the DSP has started processing Data Accumulation Interval 1, and that all events which occurred during Data Accumulation Interval 0 have been processed. The First Flag Interrupt Client function copies the completed histogram from DSP Partition 0, zeros the contents of DSP Partition 0, and notifies the thread (t15) that data is ready. This operation must complete before the DSP starts processing events from Data Accumulation Interval 2, since events accumulated during Data Accumulation Interval 2 will be accumulated into Partition 0.

When the thread is notified that data is ready, it packs and telemeters the data from Data Accumulation Interval 0 (t16). This operation must complete before the First Flag Interrupt Client function attempts to obtain the data from Data Ac-

cumulation Interval 1. Once the thread finishes processing the data, it waits again for more data.

This process continues until the thread receives a terminate command. 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.

## 4.4 Overthroughput Descriptions

Pulsar Fold Mode attempts to detect, and gracefully handle several overthroughput conditions. The following sections describe each condition, whether or not Pulsar Fold 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 process events, 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 Phase Interval. At the start of the next Phase Interval, the hardware sets a CPU readable Spillage Flag, which it clears upon the start of the subsequent Phase Interval. This flag is then read by the Interval Interrupt Client function. The Interval Interrupt Client function maintains this flag for the duration of the Data Accumulation Interval and 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.

### 4.4.2 DSP Rate exceeds Partition Rate

If the event rate into a particular histogram bin is too high, there is a possibility that the bin's count will wrap. Typically, this condition arises from an improperly configured observation. Pulsar Fold Mode provides no explicit mechanisms for detecting this condition. It is up to the ground to configure the observation such that this condition never occurs, or such that the condition can be detected by analyzing the data once it is sent to the ground.

### 4.4.3 Partition Rate exceeds First Flag Copy Rate

If the Data Accumulation Intervals are scheduled short enough, and the Partition size is set large enough, the First Flag Interrupt Client function does not have enough time to copy data from the DSP's partition and zero the partition. Outright corruption of this and subsequent partitions will occur. This condition will also cause a second First Flag Interrupt request to occur before the completion of the first. Overloaded First Flag Interrupt processing can happen due to one of two causes; the observation is improperly configured, or the system has been corrupted. This condition is detected and handled by the **Timing Module** when the

First Flag Interrupt Client function returns. If the **Timing Module** detects the occurrence, it will issue a panic and reset the EA.

### 4.4.4 First Flag Copy Rate exceeds Data Packing Rate

If the observation is configured such that the rate at which partitions 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 partitions of data. The ground can detect this condition by noticing gaps in the Partition Numbers, contained in the Partition Header Packets.

### 4.4.5 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 4.4.4.

### 4.4.6 Dropped Partition Implementation Notes

Partitions are dropped whenever the First Flag Interrupt Client function has nowhere to copy a partition of data. 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 partition's data, updates any bookkeeping variables, and zeros the DSP Partition.

## 4.5 Module Structure

### 4.5.1 Pulsar Fold Mode CPU Function Structure

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

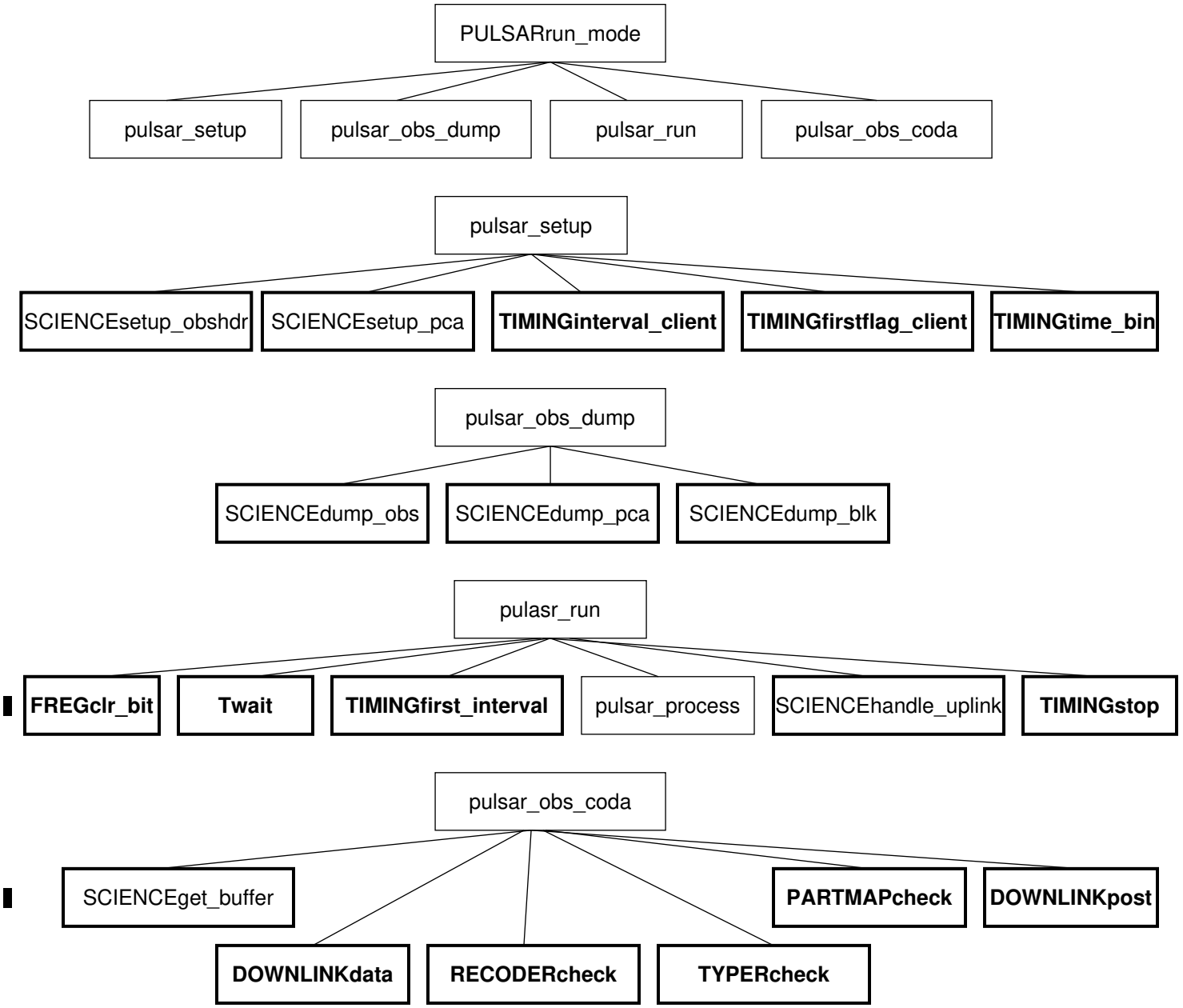


Figure 3 Pulsar Fold Mode CPU Function Structure

Figure 3 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 3. The details of each function's operation and arguments are described in Pulsar Fold Mode's detailed design document.

PULSARrun_mode ..... Run Pulsar Fold Mode Observation.
pulsar_setup ..... Setup state variables and hardware.
pulsar_obs_dump ..... Telemeter Parameter Set.
pulsar_run ..... Start and manage data acquisition.
pulsar_obs_coda ..... Check parameters and send Observation Coda.

### 4.5.2 Data Telemetry Function Structures

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

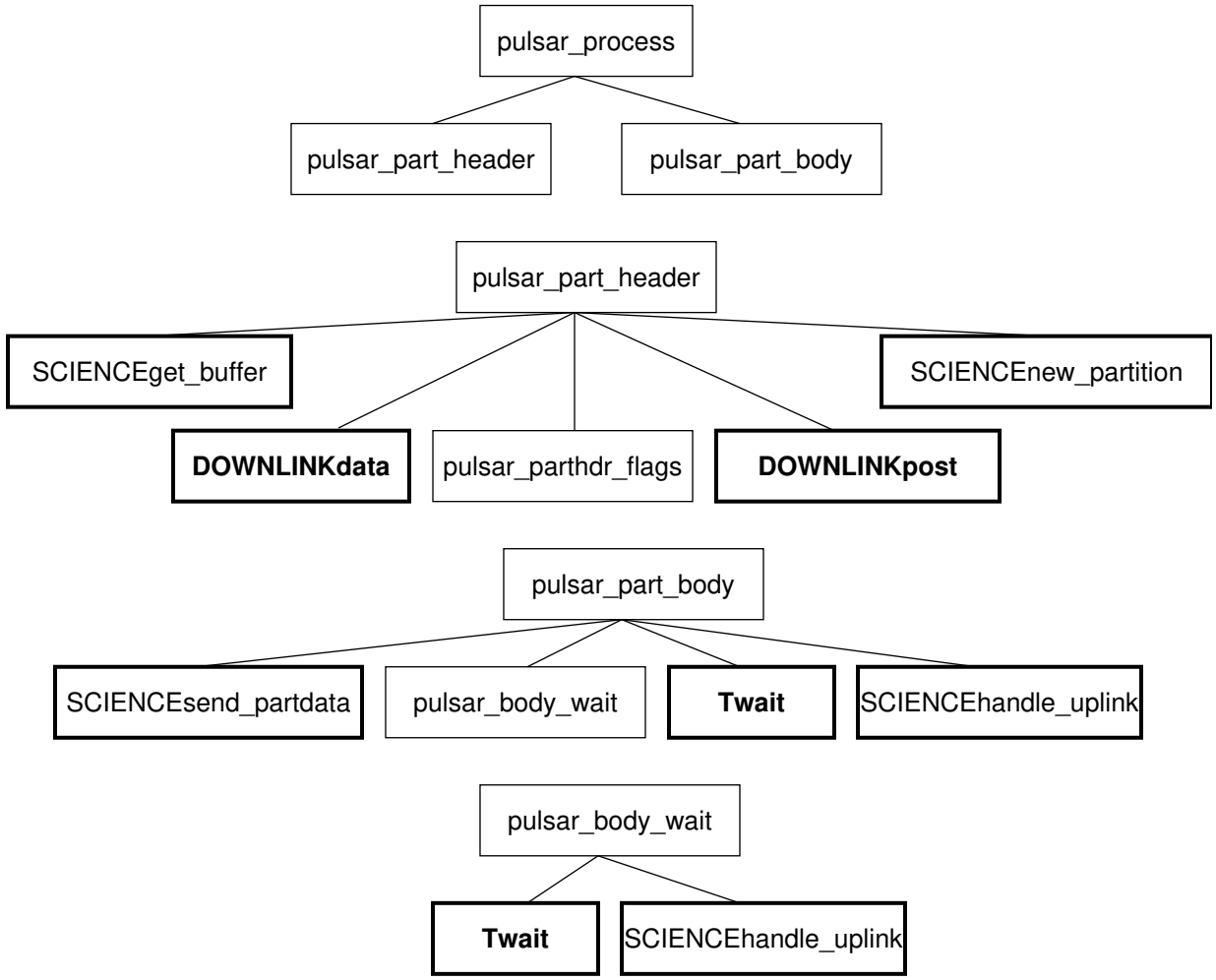


Figure 4 Data Telemetry Function Structures

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 Pulsar Fold Mode's detailed design document.

pulsar_process ..... Telemeter 1 partition of data
pulsar_part_header..... Form and telemeter a Partition Header
pulsar_parthdr_flags ..... Form the Partition Header flags.
pulsar_part_body ..... Form and telemeter the Partition Science Data
pulsar_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 Pulsar Fold Mode.

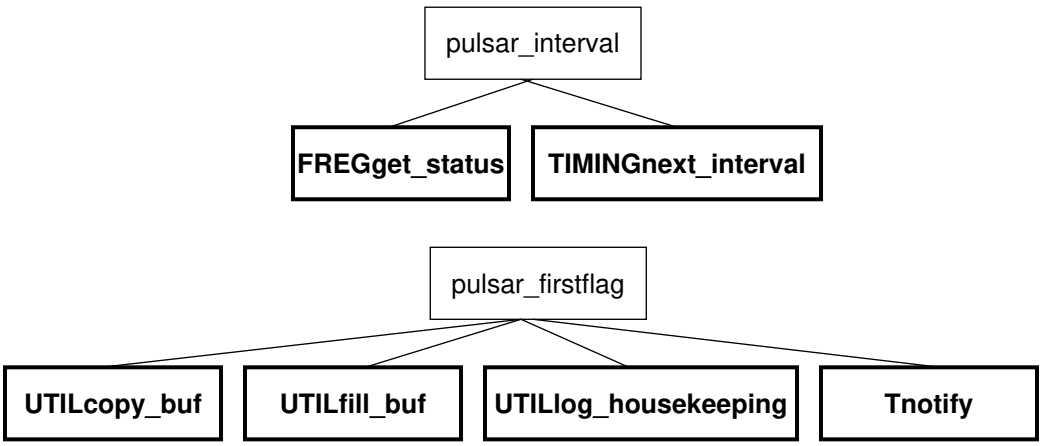


Figure 5 Interrupt Client Function Structures

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 Pulsar Fold Mode's detailed design document.

pulsar_interval ..... Schedule Intervals and maintain Interval Information.
pulsar_firstflag ..... Acquire data from DSP and reset DSP's partitions.

### 4.5.4 Phase Interval Duration Computation

This section describes how the Pulsar Fold Mode's Interval Interrupt Client function computes the duration of each Phase Interval. The desired duration of the Phase Interval's of a Data Accumulation Interval is specified in $2^{-16}$ microtick increments. Since the hardware can only support interval generation with a 1 microtick precision, Pulsar Fold Mode's Interval Interrupt Client function adjusts the duration of each Phase Interval "on the fly" to keep the total accumulated error less than plus or minus 1 microtick, relative to the specified duration. The duration

of each Phase Interval will be accurate to within 1 microtick of the specified duration, and the total error during the Data Accumulation Interval will be less than 1 microtick, for up to $2^{16}$ Phase Intervals.

The adjustment algorithm works by adjusting the duration of each Phase Interval by 1 microtick such that the total error accumulated by the current Phase Interval and by all previous Phase Intervals remains less than 1 microtick. The algorithm maintains a current accumulated error variable. This error variable represents the difference between the desired number of fractional microticks, including fractions, and the actual integer number that have been scheduled.

When scheduling a given Phase Interval, it picks either the integer interval duration, or the integer interval duration plus 1 microtick, whichever minimizes the total error accumulated up to the current Phase Interval. If the integer interval duration is picked, the added error is the fractional part of the interval duration. If the integer interval duration plus 1 microtick is picked, the added error is the fractional part of the duration minus 1 microtick. Since the fractional part is, by definition, smaller than 1 microtick, the result is a negative number. Note that the difference between the fractional part of the duration, and the fractional part minus 1 is always equal to 1. This means that, if the proper interval duration is selected, the total accumulated error, with respect to the requested duration, is always greater than or equal to -1/2 a microtick and less than or equal to 1/2 a microtick (i.e. -0.5 >= error <= 0.5).

For example, start with an initial error of 0. For this example, the fractional duration is referred to as "F." Pick the duration of the first Phase Interval as follows. If "F" is less than 1/2 a microtick, adding 1 to the interval duration will cause an error greater than 1/2 a microtick (i.e. if "F" < 0.5 then ("F" - 1) < -0.5). Use the integer interval duration, setting the error to "F." If "F" is greater than 1/2 a microtick, add 1 to the integer interval duration, and set the error to ("F" - 1) microticks. The error introduced by the first Phase Interval is greater than or equal to -1/2 a microtick and less than 1/2 a microtick.

For the subsequent Phase Intervals, first test to see if the accumulated error introduced by using the integer interval duration is less than 1/2 a microtick. Do this by adding "F" to the current accumulated error and by comparing the result to 1/2 a microtick. If the result is less than 1/2 a microtick, use the integer interval duration as is, and add "F" to the accumulated error value. If it is greater than or equal to 1/2 a microtick, use the integer interval duration plus 1, and adjust the accumulated error by adding ("F" - 1). Since "F" plus the accumulated error is greater than or equal to 1/2 a microtick, and ("F" - 1) is less than zero and greater than or equal to -1/2, the resulting accumulated error will be less than 1/2 a microtick and greater than or equal to -1/2 a microtick.

One can optimize the algorithm described in the previous paragraph by using the error variable for both keeping track of the accumulated error, and for per-

forming the test which selects which interval duration to use for the Phase Interval. Start by adding "F" to the accumulated error. If the result is less than 1/2 a microtick, use the unmodified integer interval duration. The accumulated error has already been adjusted. If the result is greater than or equal to 1/2, use the integer interval duration plus 1, and adjust the accumulated error by subtracting 1 microtick from the tested value.

### 4.5.5 Pulsar Fold Mode DSP Code Flow Chart

The DSP Code used by Pulsar Fold Mode is identical to that used by **Binned Science Mode** (see **Binned Mode Module Specification**, "Binned Mode DSP Code Flow Chart"). The following diagram illustrates the operation of the DSP when running Pulsar Fold Mode.

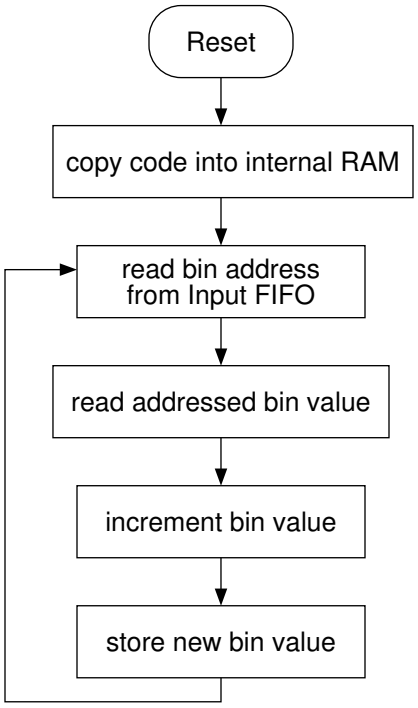


Figure 6 Pulsar Fold Mode DSP Code Flow Chart

This loop is extremely simple and fast. The DSP starts by copying the main body of the loop into its internal RAM, so that the loop can run as fast as possible. At the beginning of the loop, the DSP reads a processed event word from its Input FIFO. It then uses the read word as an address into its histogram memory. It then reads the contents of the addressed location (the bin) and increments the value. It then stores the value back into the histogram. This loop repeats until the DSP is reset.

# 5 Requirements Traceability

## 5.1 Science Requirements

Pulsar Fold Mode addresses all Level 0 and Level 1 PCA Pulsar Fold Mode Science Requirements, as specified in the **EDS-PCA Science Requirements,** "Pulsar Fold Mode". The requirements which have been addressed by this document are listed in the **Requirements Traceablity Matrix**.

The ability of Pulsar Fold 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 Pulsar Fold Mode Parameter Block extract the proper items and test the items for valid ranges.
These functions include pulsar_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. Since the bulk of the Pulsar Fold Mode's operation involves packing and posting science data, pay special attention to the time it takes for pulsar_part_data to pack and post a partition of data. Also pay special attention to the time it takes for pulsar_interval to execute, since this function governs the minimum Phase Interval duration that can

be supported, and has a direct impact on the performance of the EA while performing this mode.

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

## 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 Pulsar Fold Mode in conjunction with the other FS modules. These tests should include:

a. Verify that the Pulsar Fold Mode starts properly after a reboot.

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

c. Measure DSP performance, to determine the absolute maximum raw event rate supported by the mode.

d. Measure Data Packing and transfer performance to determine the maximum partition rate and size that can be supported.

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

f. 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 Phase Inter-

vals). This measurement determines the operational limits of the "Integer Interval Duration" parameter described in Section 2.1.

g. Use the requirements listed in the **EDS-PCA Science Requirements**, "Pulsar Fold Mode" to generate tests to verify the operation of this mode.

# 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 | | |