Back Registers Module Specification

James Francis

MIT 64-30110.0101

Rev. C

October 12, 1993

Contents

Figures

1 Introduction

1.1 Scope

This document describes the components that make up the Back Registers Software Module (BREG). This module consists of a set of functions that the Flight Software uses to manipulate some of the hardware registers accessible to the "Back End" of a given Event Analyzer (EA) in the Experimental Data System (EDS).

This document does not describe the overall design of the EDS, nor does it describe the operating details of each component in this module. It assumes that the reader is extremely familiar with the Flight Software's Global Design, with the "C" programming language and with the architecture of the EDS.

This document should be of primary interest to Flight Software Engineering, Flight Hardware Engineering, Ground Software Engineering and Software Quality Assurance.

1.2 References

MIT 64-02006.01 Rev. D .... "XTE EDS Hardware Functional Description and Requirements"
MIT 64-30102 Rev. F .... "Flight Software Global Design"
MIT 64-30110.1201 Rev. E.............. "Interrupt Manager Module Specification" (INTR)
MIT 64-30110.2301 Rev. D .... "Startup Module Specification" (STARTUP)
MIT 64-30110.0701 Rev. D .... "Timing Module Specification" (TIMING)
MIT 64-30110.2001 Rev. C .... "Utility Module Specfication" (UTIL)

1.3 Notations

This document assumes the following notations:

INT8	- Refers to an 8-bit signed value.
UINT8	- Refers to an 8-bit unsigned value.
INT16	- Refers to a 16-bit signed value.
UINT16	- Refers to a 16-bit unsigned value.
INT32	- Refers to a 32-bit signed value.
UINT32	- Refers to a 32-bit unsigned value.

1.4 Module Perspective

The following diagram is a copy of the Level 1 data flow diagram from the **Flight Software Global Design**, Section 4.1.2. The Back Registers Module does not appear directly in this data flow, but has a scope which affects items indicated by the shaded areas:

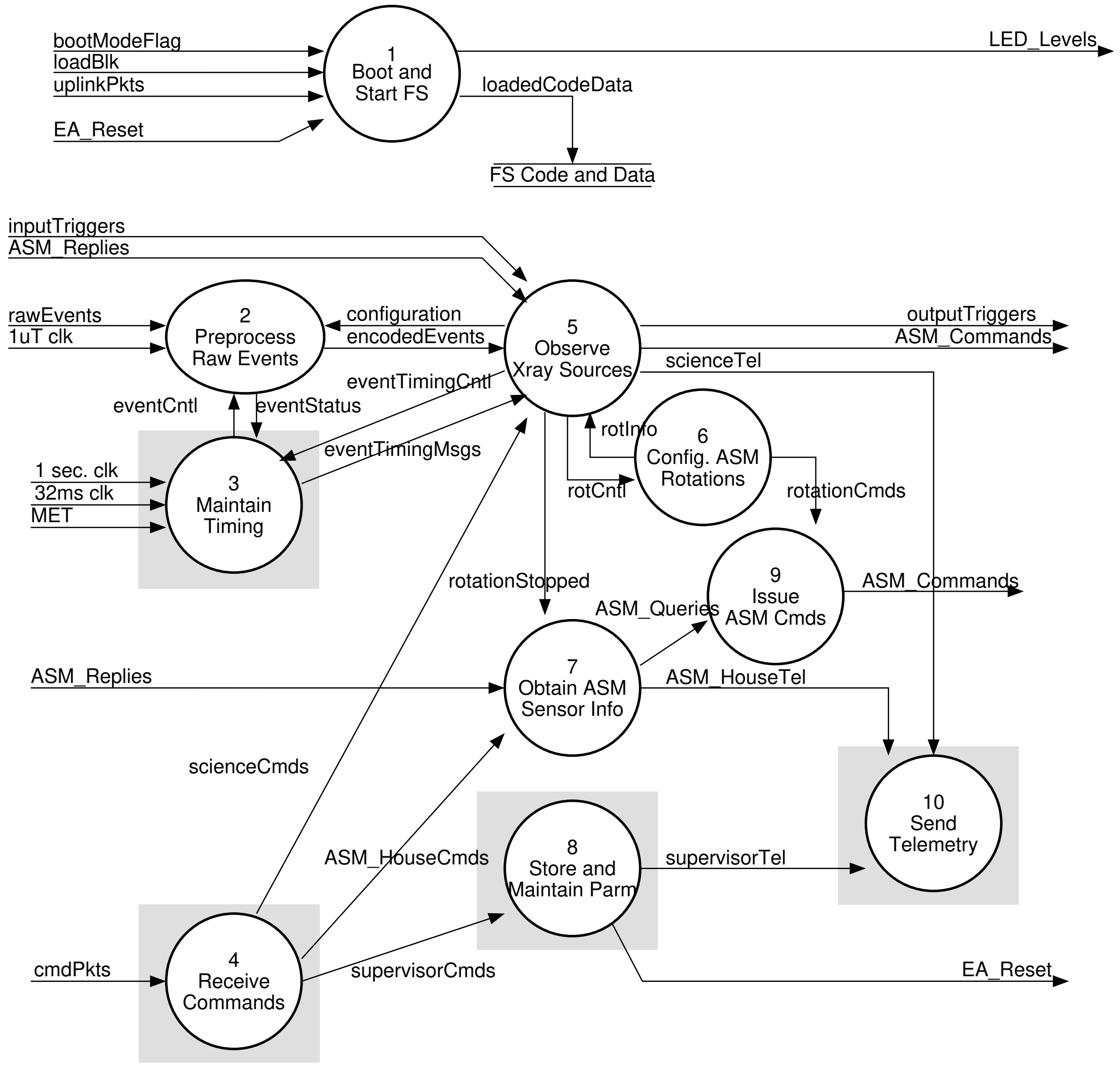

Figure 1 Module Perspective

1.5 Document Overview

Sections 1 and 2 provide an overview of the Back Registers Module, summarizing the purpose, services, context, and interfaces of the module. Section 3 describes all software structures and constants visible to modules outside of the Back Registers Module. Section 4 describes the functions provided to other modules by the Back Registers Module. Section 5 describes some of the implementation details of the module. Section 6 describes which requirements are met by the module. Section 7 describes what types of tests need to be performed to ensure that the implementation of the module is operating properly. Appendix A lists items which are currently undefined, or that need to be reviewed.

1.6 Acronyms

Please refer to the **Flight Software Global Design Specification**, Section 1.5, for a list of acronyms and abbreviations.

2 Overview

2.1 Purpose

The purpose of this module is to provide an interface between other modules in the Flight Software and the EA's back-end hardware registers. This interface consists of a set of functions and definitions, which access the CPU Control, Status and Pulse Registers, and the Mission Elapsed Time (MET) Registers.

2.2 Services

The Back Registers Module provides the following services:

1) Set bits in CPU Control Register . The Back Registers Module provides a mechanism for setting one or more bits in the CPU Control Register.
2) Clear bits in CPU Control Reg. .... The Back Registers Module provides a mechanism for clearing one or more bits in the CPU Control Register.
3) Read CPU Control Register .... The Back Registers Module provides a mechanism for reading the contents of the CPU Control Register.
3) Read CPU Status Register .... The Back Registers Module provides a mechanism for reading the contents of the CPU Status Register.
4) Assert CPU Pulse Register .... The Back Registers Module provides a mechanism for asserting one or more bits in the CPU Pulse Register.
5) Obtain the EA Id .... The Back Registers Module provides a mechanism for obtaining the Event Analyzer Id from the CPU Status Register.
6) Obtain the MET .... The Back Registers Module provides a mechanism for obtaining the contents of the Mission Elapsed Time (MET) Registers.

2.3 Module Context

The following illustrates the Back Registers Module's relationships with the rest of the Flight Software residing in an EA:

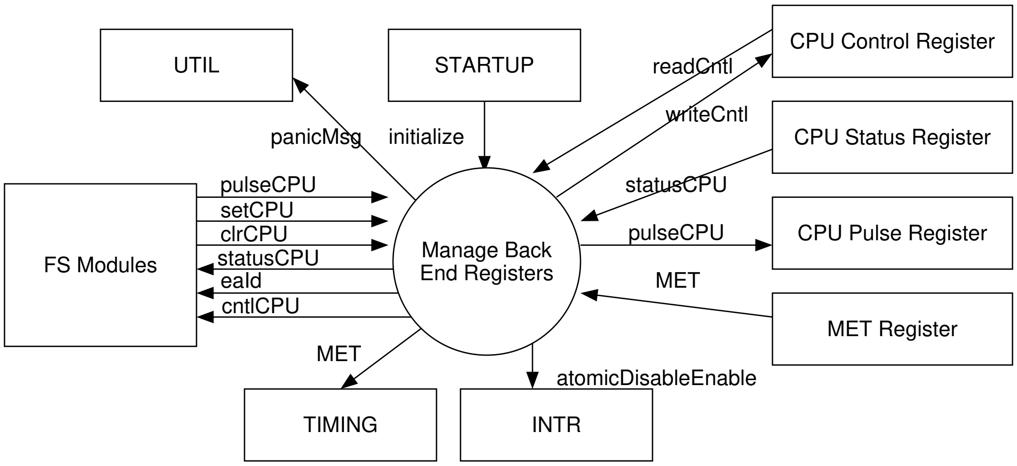

Figure 2 Back Registers Module Context Diagram

The Back Registers Module reads and writes directly to and from the CPU Control Register (writeCntl, readCntl), reads directly from the CPU Status Register (statusCPU), writes directly to the CPU Pulse Register (pulseCPU) and reads directly from the Mission Elapsed Time (MET) register.

During system initialization (initialize), the Back Registers Module deasserts all control signals in the CPU Control Register by writing directly to the CPU Control Register (writeCntl). It resets the MajorTick latch and the MinorTick Interrupt latch by writing directly to the CPU Pulse Register (pulseCPU). The Back Registers Module does not reset the Uplink Interrupt Latch. This allows the Flight Software to detect if a command was received and lost during boot.

Once running, the **Timing Module** can obtain the MET value (MET) using the Back Registers Module. It is the **Timing Module**'s responsibility to ensure that the register is read no earlier than 100ms before the next MajorTick pulse and no later than 100ms after the pulse (see **Timing Module Specification**, TIMINGtrigger_hexte).

The Back Registers Module provides CPU Control Register bit manipulation services to the rest of the Flight Software (setCPU, clrCPU). When a bit is set or cleared, the Back Registers Module atomically reads the CPU Control Register, performs the bit operation, and writes the value back to the Control Register. It uses the **Interrupt Manager Module** to perform the interrupt disable and enables (atomicDisableEnable).

The Back Registers Module also provides general Flight Software access to the CPU Control Register (cntlCPU), the CPU Status Register (statusCPU) and CPU Pulse Register (pulseCPU). In addition to these services, it provides an extracted version of the Event Analyzer Id (eaId) read from the CPU Status Register.

If the Back Registers Module encounters an error or inconsistency, it uses the **Utility Module** to issue a panic message and reset the EA (panicMsg).

2.4 Data Structures and Constants Summary

The CPU Control, Status, Pulse, and MET hardware registers define all constants and bit-positions that this module sees. Please refer to the **XTE EDS Hardware Functional Description and Requirements** for a detailed description of these registers. The registers have the following meanings:

CPU Control Register	Reads and Writes Self Interrupt Requests, Downlink Requests, and LED bits.
CPU Status Register	Contains MinorTick and MajorTick Timing Latch Information, EA Id, Watchdog activation flag and the Uplink Boot Flag.
CPU Pulse Register	Used to reset the MinorTick and MajorTick latches, and start and reset the Watchdog timer.
MET Registers	These are two 16-bit registers which contain the current SDS supplied Mission Elapsed Time value.

2.5 Function List Summary

This interface defines the following functions:

BREGinit()	Initializes the state of this Module
BREGget_control()	Returns contents of the control register
BREGset_bit()	Sets bits in the control register
BREGclr_bit()	Clears bits in the control register
BREGget_status()	Returns contents of the status register
BREGpulse()	"Pulses" bits in the pulse register
BREGread_met()	Returns contents of the MET registers
BREGea_id()	Returns the Event Analyzer Id

3 Data Structures and Constants

The following is a summary of the registers and the current bit-definitions of each register. All bit definitions are active high. For more detail, refer to the **XTE EDS Hardware Functional Description and Requirements**.

3.1 CPU Control Register (16-bit 0x090000 flat-address r/w)

CPU_CNTL_SELF_INT .... (0x00001).................. Cause Self Interrupt
CPU_CNTL_DNLK_RQST............. (0x00002).................. Issue Downlink Request to Send
Unused .... (0x00004).................. Bit 2 is unused
CPU_CNTL_LED_0 .... (0x00008).................. LED Bit 0 (0->OFF, 1->ON)
CPU_CNTL_LED_1 .... (0x00010).................. LED Bit 1
CPU_CNTL_LED_2 .... (0x00020).................. LED Bit 2
Bits 6-15 are unused.

3.2 CPU Status Register (16-bit 0x0a0000 read only)

CPU_STAT_MIN_LAT .... (0x00001).................. MinorTick Interrupt Latch
CPU_STAT_WDOG_ACT .... (0x00002).................. Watchdog Timer Active
CPU_STAT_MAJ_LAT................... (0x00004).................. MajorTick Pulse Latch
CPU_STAT_UPINT......................... (0x00008).................. Uplink Interrupt Latch
CPU_STAT_DNINT .... (0x00010).................. Downlink Interrupt Latch
CPU_STAT_EAID .... Bits 5 - 7 inclusive .... EA Hardwired Slot Id
CPU_STAT_BOOT_UPLK .... (0x00100).................. Boot from Uplink Flag
Bits 9-15 are unused.

3.3 CPU Pulse Register (16-bit 0x0a0000 write only)

CPU_PULSE_RST_MAJMIN .... (0x00001).................. Reset MinorTick/MajorTick Latches
CPU_PULSE_PUL_WDOG .... (0x00002).................. Start/Reset the Watchdog Timer.
Unused .... (0x00004).................. Bit 2 is unused
CPU_PULSE_RST_UPINT .... (0x00008).................. Reset Uplink Interrupt Latch
CPU_PULSE_RST_DNINT .... (0x00010).................. Reset Downlink Interrupt Latch
Bits 5-15 are unused.

3.4 MET Registers

The lower 16-bit and upper 16-bit words of the SDS supplied MET value are stored in two CPU readable registers.

3.4.1 MET Low Word Register (16-bit, 0x080000 read only)

The lower 16-bit word of the SDS supplied MET value is stored in this read only register. This register encompasses bits 0-15 of the MET value.

3.4.2 MET High Word Register (16-bit, 0x080002 read only)

The upper 16-bit word of the SDS supplied MET value is stored in this read only register. This register encompasses bits 16-31 of the MET value.

3.5 Event Analyzer Id Assignments

The following lists each Event Analyzer in the EDS and their id assignments:

CPU_EAID_A_ASM .... (0)............................. Subgroup A, ASM-EA
CPU_EAID_A_PCA1 .... (1)............................. Subgroup A, PCA-EA 1
CPU_EAID_A_PCA2 .... (2)............................. Subgroup A, PCA-EA 2
CPU_EAID_A_PCA3 .... (3)............................. Subgroup A, PCA-EA 3
CPU_EAID_B_ASM....................... (4)............................. Subgroup B, ASM-EA
CPU_EAID_B_PCA1 .... (5)............................. Subgroup B, PCA-EA 1
CPU_EAID_B_PCA2 .... (6)............................. Subgroup B, PCA-EA 2
CPU_EAID_B_PCA3 .... (7)............................. Subgroup B, PCA-EA 3

3.6 Constants

3.6.1 CPU_CNTL_UNUSED_MASK (0x0ffc4)

This mask identifies the unused bits in the CPU Control Register.

3.6.2 CPU_STAT_UNUSED_MASK (0x0fe00)

This mask identifies the unused bits in the CPU Status Register.

3.6.3 CPU_PULSE_UNUSED_MASK (0x0ffe4)

This mask identifies the unused bits in the CPU Pulse Register.

3.6.4 MSG_BREG_SET_BIT_UNUSED

This message code is passed to UTILpanic() if BREGset_bit() is passed a mask which asks to set an unused bit in the CPU Control Register. The message should read something like:

"PANIC: ---BREGset_bit(%d) --Request to set unused bit"
where %d represents the passed mask argument.

3.6.5 MSG_BREG_SET_BIT_ZERO

This message code is passed to UTILpanic() if BREGset_bit() is passed a zero valued mask. The message should read something like:

"PANIC: ---BREGset_bit() --Zero mask"

3.6.6 MSG_BREG_CLR_BIT_UNUSED

This message code is passed to UTILpanic() if BREGclr_bit() is passed a mask which asks to clear an unused bit in the CPU Control Register. The message should read something like:

"PANIC: ---BREGclr_bit(%d) --Request to clear unused bit"
where %d represents the passed mask argument.

3.6.7 MSG_BREG_CLR_BIT_ZERO

This message code is passed to UTILpanic() if BREGclr_bit() is passed a zero valued mask. The message should read something like:

"PANIC: ---BREGclr_bit() --Zero mask"

3.6.8 MSG_BREG_PULSE_BIT_UNUSED

This message code is passed to UTILpanic() if BREGpulse() is passed a mask which asks to pulse an unused bit in the CPU Pulse Register. The message should read something like:

"PANIC: ---BREGpulse(%d) --Request to pulse unused bit"
where %d represents the passed mask argument.

3.6.9 MSG_BREG_PULSE_ZERO

This message code is passed to UTILpanic() if BREGpulse() is passed a zero valued mask. The message should read something like:

"PANIC: ---BREGpulse() --Zero mask"

4 Functions

4.1 BREGinit() - Initialize Module

Usage

void BREGinit (void);

Description

BREGinit() initializes the Back Registers module. It initializes module specific data and deasserts all signals in the Control Register (such as the Self Interrupt Request and Downlink Request to Send signals). It also resets the MinorTick Interrupt and MajorTick pulse latches and the Downlink Interrupt latch using the Pulse Register. BREGinit() does not reset the Uplink Interrupt latch, thus allowing the Flight Software to detect a dropped command, received during the boot process.

Interrupts MUST be disabled before calling this function. This function MUST be called before using any other functions in this module.

Return Values

NONE - FUNCTION IS VOID.

Caveats

1) This function should only be called during system initialization.

See Also

NONE

4.2 BREGget_control() - Read Control Register

Usage

UINT16 BREGget_control (void);

Description

BREGget_control() returns the contents of the Back End's CPU Control Register. This function sets all undefined bits in the returned value to one. BREGinit() must be called before using this function. The meaning of the return value is described in Section 3.1.

Return Values

unsigned, 16-bit contents of the CPU control register.

Caveats

1) BREGinit() MUST be called before using this function.

See Also

BREGset_bit(), BREGclr_bit().

4.3 BREGset_bit() - Set Bits in Control Register

Usage

void BREGset_bit (UINT16 *mask*);

Description

BREGset_bit() sets the bits in the control register as specified by the "*mask*" argument. Each '1' bit in "*mask*" causes this function to set the corresponding bit in the control register to a '1'. All other bits are unaffected. Bits may be cleared using another function, BREGclr_bit(). The result may be examined by calling BREGget_control(). BREGinit() must be called before using this function. See Section 3.1 for a description of this register.

If "*mask*" specifies a bit which is undefined in the CPU Control Register (see CPU_CNTL_UNUSED_MASK in Section 3.6.1), BREGset_bit() will issue a MSG_BREG_SET_BIT_UNUSED panic message, passing "*mask*" as the message argument, using the **Utility Module** function UTILpanic().

If "*mask*" is zero, BREGset_bit() will issue a MSG_BREG_SET_BIT_ZERO panic message and reset the EA, using the **Utility Module** function UTILpanic().

Return Values

NONE - FUNCTION IS VOID

Caveats

1) BREGinit() MUST be called before using this function

See Also

BREGget_control(), BREGclr_bit()

Utility Module Specification

4.4 BREGclr_bit() - Clear Bits in Control Register

Usage

void BREGclr_bit (UINT16 *mask*);

Description

BREGclr_bit() clears the bits in the control register as specified by the "*mask*" argument. Each '1' bit in "*mask*" causes this function to clear the corresponding bit in the control register to a '0'. All other bits are unaffected. Bits may be set by using another function, BREGset_bit(). The result may be examined by calling BREGget_control(). BREGinit() must be called before using this function. See Section 3.1 for a description of this register.

If "*mask*" specifies a bit which is undefined in the CPU Control Register (see CPU_CNTL_UNUSED_MASK in Section 3.6.1), BREGclr_bit() will issue a MSG_BREG_CLR_BIT_UNUSED panic message, passing "*mask*" as the message argument, using the **Utility Module** function UTILpanic().

If "*mask*" is zero, BREGclr_bit() will issue a MSG_BREG_CLR_BIT_ZERO panic message and reset the EA, using the **Utility Module** function UTILpanic().

Return Values

NONE - FUNCTION IS VOID

Caveats

1) BREGinit() MUST be called before using this function.

See Also

BREGget_control(), BREGset_bit(),

Utility Module Specification

4.5 BREGget_status() - Read Status Register

Usage

UINT16 BREGget_status (void);

Description

BREGget_status() reads the current contents of the Back End's CPU Status Register. This function sets all undefined bits in the return value to one. BREGinit() must be called before using this function. This register is READ-ONLY, so there is no function to explicitly set the value of this register. See Section 3.2 for a description of this register.

Return Values

unsigned, 16-bit contents of the CPU Status Register

Caveats

1) BREGinit() MUST be called before using this function.

See Also

NONE

4.6 BREGpulse() - Pulse Bits in Pulse Register

Usage

void BREGpulse (UINT16 *mask*);

Description

BREGpulse() pulses the bits in the CPU Pulse Register as specified by the argument "*mask*". Each "1" bit in "*mask*" causes the corresponding bit in the register to be pulsed. BREGinit() must be called before using this function. See Section 3.3 for a description of this register.

If "*mask*" specifies a bit which is undefined in the CPU Pulse Register (see CPU_PULSE_UNUSED_MASK in Section 3.6.3), BREGpulse() will issue a MSG_BREG_PULSE_BIT_UNUSED panic message, passing "*mask*" as the message argument, using the **Utility Module** function UTILpanic().

If "*mask*" is zero, BREGpulse() will issue a MSG_BREG_PULSE_ZERO panic message and reset the EA, using the **Utility Module** function UTILpanic().

Return Values

NONE - FUNCTION IS VOID

Caveats

1) BREGinit() MUST be called before using this function.

See Also

Utility Module Specification

4.7 BREGread_met() - Read the MET Register

Usage

UINT32 BREGread_met (void);

Description

BREGread_met() reads the contents of the MET registers and returns the concatenated 32-bit result. See Section 3.4 for a description of these registers.

Return Values

NONE - FUNCTION IS VOID

Caveats

1) BREGinit() MUST be called before using this function.

2) To obtain a valid result, this function must be called within 100ms of a MajorTick pulse.

See Also

Timing Module Specification

4.8 BREGea_id() - Obtain the Event Analyzer Id

Usage

UINT8 BREGea_id (void);

Description

BREGea_id() reads the contents of the CPU Status Register and extracts the Event Analyzer Id.

Return Values

The Event Analyzer's Id
See Section 3.5 for a list of the EA Id assignements.

Caveats

1) BREGinit() MUST be called before using this function.

See Also

NONE

5 Detailed Design Considerations

5.1 Data Flow

The following diagram illustrates the level 1 data flow within the Back Registers Module:

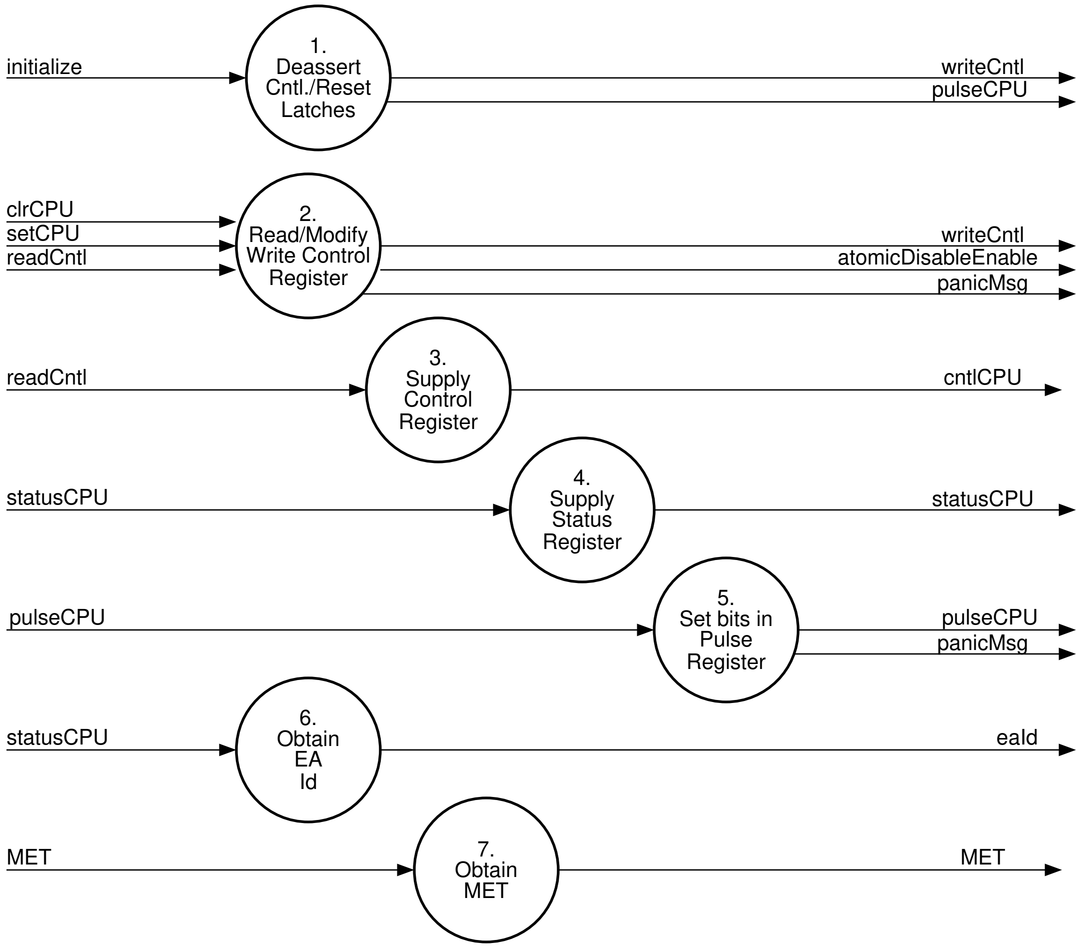

Figure 3 Back Registers Level 1 Data Flow

During EA initialization (bubble 1), the Back Registers Module deasserts all signals in the CPU Control Register (writeCntl), such as the Self Interrupt Request and Downlink Request to Send signals, and resets the latched versions of the

MinorTick Interrupt, MajorTick pulse, Downlink Interrupt and Uplink Interrupt (pulseCPU).

The Back Registers Module provides the capability to set or clear specific bits in the CPU Control Register (bubble 2). In one atomic step (atomicDisableEnable), it reads the CPU Control Register (readCntl), sets (setCPU) or clears (clrCPU) the specified bits, and writes the modified version back to the CPU Control Register (writeCntl). If a request is made to set or clear an unused bit in the CPU Control Register, the Back Registers Module emits a panic message and resets the EA (panicMsg).

When requested to supply the CPU Control Register (bubble 3), the Back Registers Module obtains the CPU Control Register (readCntl) and returns the read version to the caller (cntlCPU).

When requested to supply the CPU Status Register (bubble 4), the Back Registers Module reads the status register and returns it to the caller (statusCPU).

When requested to assert bits in the CPU Pulse Register (bubble 5), the Back Registers Module writes the request directly to the CPU Pulse Register (pulseCPU). If a request is made to pulse an unused bit in the CPU Pulse Register, the Back Registers Module emits a panic message and resets the EA (panicMsg).

When requested to supply the Event Analyzer Id (bubble 6), the Back Registers Module reads the CPU Status Register (statusCPU). It then masks the read value and shifts the result so that the 3 EAId bits are in the lower 3 bits of the returned value (eaId).

When requested to supply the MET value (bubble 7), the Back Registers Module reads the contents of the MET registers and concatenates and returns the formed value (MET).

5.2 Control Considerations

5.2.1 CPU Control Register Bit Manipulations

Since the Back Registers Module sets and clears bits in the CPU Control Register by reading the current settings, modifying, and then writing the new settings, it must ensure that the register isn't modified between the time it is read and the time it is written. The Back Registers Module does this by using the **Interrupt Manager Module** to ensure that interrupts are disabled while performing the operation.

5.3 Module Structure

This section describes the structure of the Back Registers Module's functions. The following diagram illustrates this structure, where bolded boxes and text indicate functions provided by modules other than the Back Registers Module.

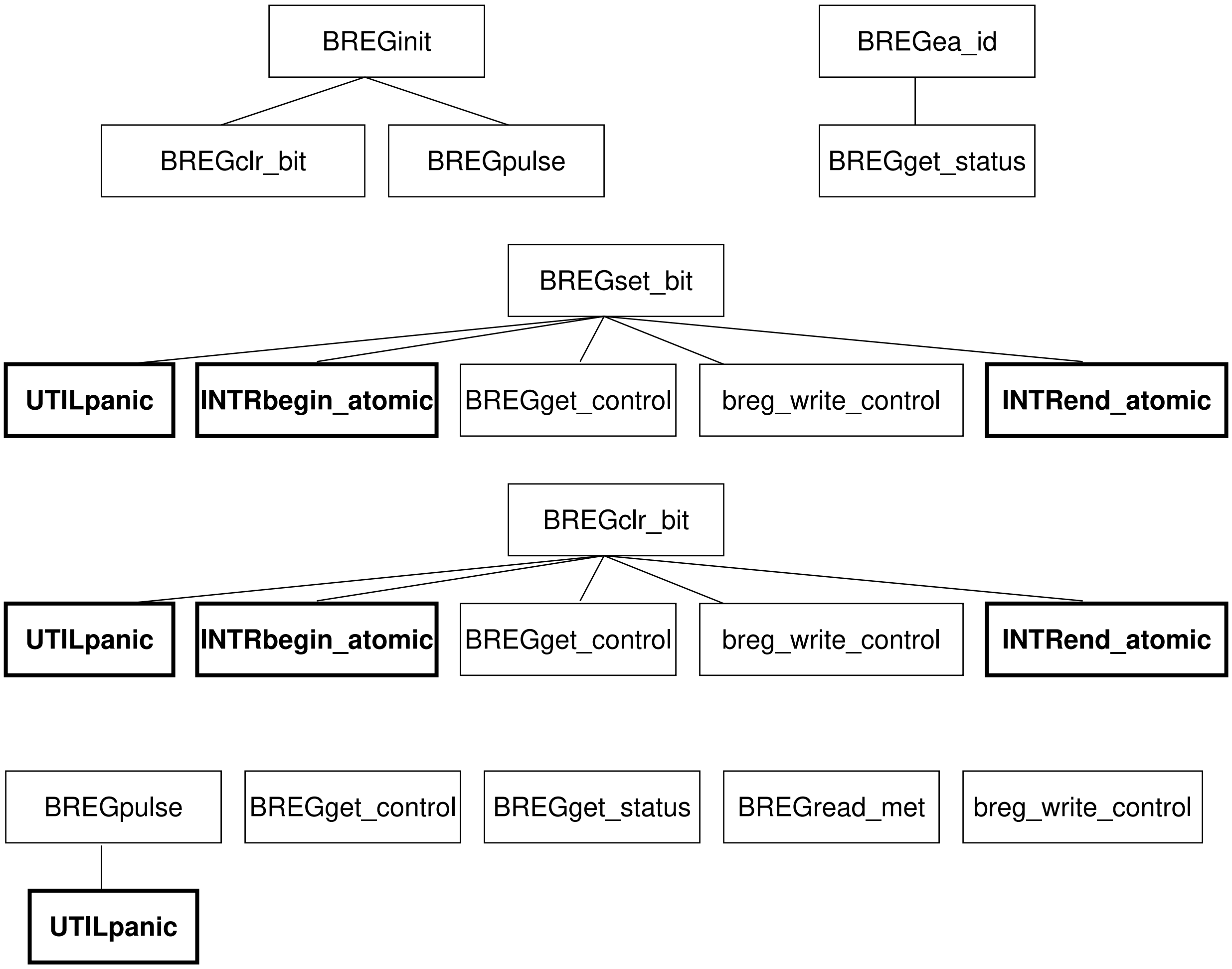

Figure 4 Back Registers Function Structures

The following describes each of the functions shown in Figure 4. The details of each function's operation and arguments are described in the Back Registers Module's detailed design document.

BREGinit .... This function is invoked by the **Startup Module**, before any other functions in this module are called. This function deasserts all control signals in the CPU Control Register, such as the Self Interrupt Request and Downlink Request to Send signals, using BREGclr_bit. It then resets the MinorTick Interrupt, MajorTick, and Downlink Interrupt Interrupt latches, using BREGpulse.

BREGea_id .... This function is called by the Flight Software Modules to obtain the Event Analyzer Id. BREGea_id uses BREG-get_status to obtain the CPU Status Register, "and"'s the returned value to retain only the EA Id bits, and shifts the result so that the EA Id is in the lower three bits. It then returns the result.

BREGset_bit .... This function is called by the Flight Software Modules to set one or more bits in the CPU Control Register to one. BREGset_bit performs this operation as an atomic step (interrupts disabled) using the **Interrupt Manager Module** (INTRbegin_atomic, INTRend_atomic). It reads the current CPU Control Register, using BREGget_control, "or"'s it with the mask passed by the caller, and writes the new value using a local function, breg_write_control. If a request is made to set an unused bit in the CPU Control Register, or if the passed mask is zero, BREGset_bit issues a panic message and resets the EA, using the **Utility Module** (UTILpanic).

BREGclr_bit .... This function is called by the Flight Software Modules and by BREGinit to set one or more bits in the CPU Control Register to zero. BREGclr_bit performs this operation as an atomic step (interrupts disabled) using the **Interrupt Manager Module** (INTRbegin_atomic, INTRend_atomic). It reads the current CPU Control Register, using BREGget_control, inverts the bits in the mask passed by the caller, "and"'s the two together, and writes the new value using a local function, breg_write_control. If a request is made to clear an unused bit in the CPU Control Register, or if the passed mask is zero, BREG-clr_bit issues a panic message and resets the EA, using the **Utility Module** (UTILpanic).

BREGget_control .... This function is called by the Flight Software Modules and by BREGset_bit and BREGclr_bit to obtain the current value of the CPU Control Register. This function reads the CPU Control Register and returns the read value to the caller.

BREGget_status................................. This function is called by the Flight Software Modules, and by BREGea_id to obtain the current value of the CPU Status Register. This function reads the CPU Status Register and returns the read value to the caller.

BREGpulse .... This function is called by the Flight Software Modules, and by BREGinit to set one or more bits in the CPU Pulse Register. This function writes the value passed by the caller directly into the CPU Pulse Register. If a request is made to pulse an unused bit in the CPU Pulse Register, or if the passed mask is zero, BREGpulse issues a panic message and resets the EA, using the **Utility Module** (UTILpanic).

BREGread_met .... This function is called by the **Timing Module** to obtain the current MET value. This function reads lower and upper words of the MET value, combines the read values and returns the result to the caller. It is up to the **Timing Module** to ensure that this function is called no sooner than 100ms before a MajorTick pulse and no later than 100ms after a MajorTick pulse.

breg_write_control .... This function is private to the Back Registers Module. It is called by BREGset_bit and BREGclr_bit to write a new value into the CPU Control Register. This function writes the passed value directly into the register.

6 Requirements Traceability

6.1 Science Requirements

This module does not address any specific Science Requirements, beyond the ability to access the Back End’s Control, Status, Pulse and MET registers.

7 Testing Considerations

7.1 Unit Testing

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

a. An Event Analyzer equipped with the Intermetrics ROM Monitor

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 verify that the hardware is configured and programmed correctly by all functions which affect the CPU Control and Pulse Registers, and that the CPU Status and MET Registers are read and interpreted correctly.

7.2 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. Receive telemetry packets via the RS422 Interface (S/C Simulator)

   2. Supply an incrementing MET value (S/C Simulator)

   3. Supply the MinorTick and MajorTick clocks (S/C Simulator)

The integration tests should verify the operation of the Back Registers Module in conjunction with the actual hardware and other FS modules. These tests should include:

a. Verify that the Downlink Request to Send line is asserted correctly.

b. Verify that MET updates are read and intepreted correctly.

c. Verify that the MinorTick, Uplink and Downlink interrupt latches are properly reset.

d. Verify that a lost command can be detected during an EA boot.

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		