The GLAST experiment at SLAC

The ACD Electronics Module (AEM)

Electronics group

A Primer

Document Version:	2.2
Document Issue:	1
Document Edition:	English
Document Status:	Draft - for internal distribution only
Document ID:	LAT-TD-00639-D1
Document Date:	January 14, 2003

Stanford Linear Accelerator Center (SLAC)
2575 Sandhill Road
Menlo Park California, 94025 USA

Abstract

A description, written from the viewpoint of *use* of the AEM (ACD Electronics Module). This includes:

- naming conventions between ACD and AEM
- enumeration of the AEM's registers
- the protocol used to access the AEM's registers and functional blocks
- a description of the event data emitted by the AEM

Intended audience

This document is intended principally as a guide for the *users* of the ACD Electronics Module (AEM). These include:

- developers of the sub-system electronics which interface with the AEM
- developers of Flight-Software
- developers of I&T (Integration and Test) based systems

All readers of this document are expected to be familiar with the concepts described in [1]

Conventions used in this document

Certain special typographical conventions are used in this document. They are documented here for the convenience of the reader:

- Any words or expressions italicized and bolded (*e.g.*, ***Event***) are intended to introduce a concept, definition, or keyword. Such words or expressions are to be found in the index.
- Acronyms are called out in a particular style (e.g., SLAC).
- Any word or expression called in code style indicates a hardware signal or register name (e.g., `RIGHT_FIRST`, or `LAYER_MASK_1`)

References

1 LAT-TD-00606-D1, “LAT Inter-module Communications - A reference manual”, by *Michael Huffer*

2 LAT-TD-0035-01, “LAT Coordinate System” 11/20/00 by *Steve Ritz*

3 LAT-SS-00363-01, “ACD Front-End Electronics to ACD Electronics Module - Interface Control document (Draft 10)” 3/15/02

4 Data sheet for the “MAXIM 145 2.7V, Low-Power, 2-channel 108ksps Serial 12-Bit ADCs in 8-Pin uMAX”. hhttp://www.maxim-ic.com

Document Control Sheet

Table 1 Document Control Sheet

Document	**Title:**	The ACD Electronics Module (AEM) A Primer		
	Version:	2.2		
	Issue:	1		
	Edition:	English		
	ID:	LAT-TD-00639-D1		
	Status:	Draft - for internal distribution only		
	Created:	February 9, 2002		
	Date:	January 14, 2003		
	Access:	\\ZWINSAN2\users6\m\mehsys\Private\glast\AEM\frontmatter.fm		
	Keywords:	template, framemaker, software documentation, document preperation, sample material		
Tools	**DTP System:**	Adobe FrameMaker	**Version:**	6.0
	Layout Template:	Software Documentation Layout Templates	**Version:**	V2.0 - 5 July 1999
	Content Template:	--	**Version:**	--
Authorship	**Coordinator:**	Michael Huffer		
	Written by:	Michael Huffer		
	Reviewed by:	N/A		
	Approved by:	N/A		

Document Status Sheet

Table 2 Document Status Sheet

Title:	The ACD Electronics Module (AEM) A Primer		
ID:	LAT-TD-00639-D1		
Version	**Issue**	**Date**	**Reason for change**
1.0	1	3/12/2002	Initial draft
1.1	1	3/27/2002	Initial draft for public comment
1.2	1	3/29/2002	Comments from GXH
1.3	1	4/01/2002	Comments from JJ
1.4	1	4/12/2002	Miscellaneous typos as mentioned by Curt and Selim in e-mails of April 3 and April 4
2.0	1	5/29/2002	Many changes given that the AEM design is actually under way, we've commissioned the VAEM and we've seen and assimilated the ACD interface through testing of the "bud box": 1. Given AEM constraints and event structure from ACD, the format of an event must be completely rethought. The event chapter has been rewritten to reflect my current thinking. *Note*: undoubtedly, as the AEM design matures, this will undergo more changes. 2. Table 10 (function block 5) numbered incorrectly. *Note*, this changes the register numbering. 3. Even read commands require a payload. Updated document accordingly. 4. Miscellaneous typos, mentioned by Curt corrected.
2.1	1	6/07/2002	Still had function block 5 numbered incorrectly. Also fixed miscellaneous typos in event chapter.
2.2	1	1/14/2003	In anticipation of V2 test-stand for ACD: 1. cleaned up notation and syntax in register chapter 2. Added power management register 3. Added new functional block to support environmental monitoring.

Table of Contents

List of Tables

List of Figures

Chapter 1 Registers

1.1 Overview

It is convenient to consider the AEM, its functional blocks types and their registers as organized hierarchically as illustrated in Figure 1. Each type, of course, may have more then one instance. For example, the AEM has *twelve* GLAST ACD Readout Controllers (GARC). The number of instances for each type is set out in Table 3.

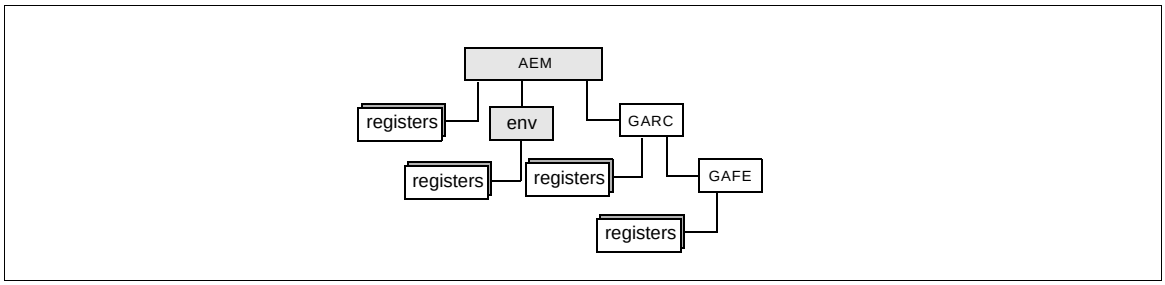

Figure 1 Hierarchy of target types

This hierarchy expresses itself in at least two usages:

Commanding: In order to access the registers or functional block. For example to either read or write a register one must specify a hierarchical path to the register. Thus, to access a register in an ACD Front-End ASIC, one must go through a AEM, GARC and GAFE. The protocol of commanding is described in Chapter 2.

Resets: `Resets` are propagated downwards from the point they are issued. For example, if the AEM is reset, it will also cause a reset of all its 18 FREE boards. A reset of a FREE board (through its GARC) will cause a reset of its Front-Ends and so forth.

Table 3 The AEM's blocks and registers

block type	# of blocks	# of registers	description
AEM	1	6	AEM Common Controller FPGA
environmental monitor	1	12	FREE board environmental monitoring
GARC	12	43 *x* 12 = 516	ACD Readout Controller ASIC
GAFE	18 *x* 12 = 216	11 *x* 216 = 2376	ACD Front-End ASIC
Total	**229**	**2910**	

1.2 Common conventions for on board registers

In describing the registers *on-board* the AEM the following conventions are used or assumed when describing a register's fields:

Not defined: Undefined fields are identified as Must Be Zero (MBZ) and are illustrated *greyed out*. An MBZ field will:

- read back as zero
- ignore writes
- reset to zero

Read/Write: A *Reset* will set a Read / Write field to zero.

Read Only: Read only fields are illustrated *lightly* greyed-out with their value. Any *Read-only* field will:

- ignore writes
- reset to zero, unless otherwise documented

Any field used as a boolean has a width of one bit. A value of one (1) is used to indicate its *set* or *true* sense and a value of zero (0) to indicate its *clear* or *false* sense. Field numbering for registers is such that offset zero (0) corresponds to a register's Least Significant Bit (LSB) and the largest offset corresponds to a register's Most Significant Bit (MSB).

1.3 The AEM (Common Controller)

This section incorporates all the registers which are in common on the AEM. As all these registers are contained physically on a single FPGA called the ***Common Controller***, AEM registers may also be referenced as *Common Controller* registers. All registers of this block are thirty-two (32) bits in length.

Table 4 The AEM *registers*

Name	Number	Access	Description
CONFIGURATION	0	R/W	Configuration and setup
COMMON_STATUS	1	R/W[1]	CSR latched values (for AEM)
CABLE_STATUS	2	R/W[1]	CSR latched cable values
COMMAND_RESPONSE	3	R/W[1]	Command/response statistics
TRGSEQ	4	R/W	Trigger sequencing
POWER_MANAGMENT	5	R/W	Power management for FREE boards
Total	**6**		

1. On write, the value is ignored and the register is set to zero (0)

1.3.1 Configuration register

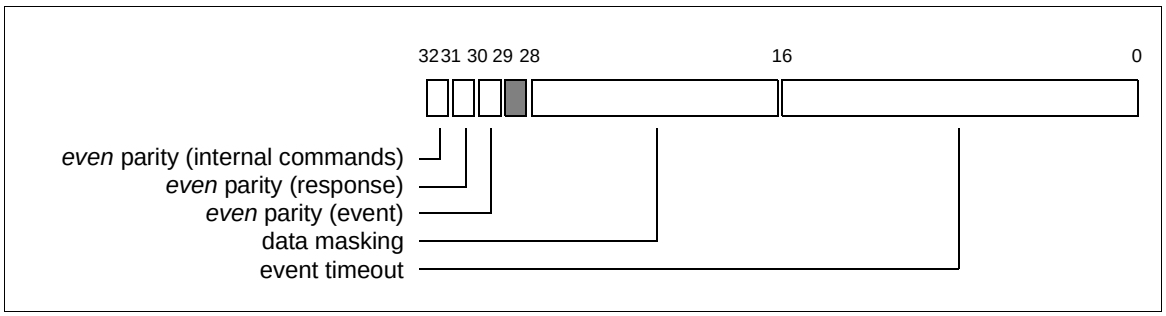

Figure 2 AEM Configuration register

even parity (internal commands): Determines whether odd or even parity is generated for any of the internal issued commands to the ACD Front-End-Electronics. These include both the CALSTROBE and TACK commands. If this field is *clear*, *odd* parity will be generated. If this field is *set*, *even* parity will be generated. The value of this field lies only in its use as a debugging and commissioning tool to purposely stimulate parity errors at the *destination* of these commands.

even parity (response): Determines whether odd or even parity is generated for the message. corresponding to a command request response. If this field is *clear*, *odd* parity will be generated. If this field is *set*, *even* parity will be generated. The value of this field lies only in its use as a debugging and commissioning tool to purposely stimulate parity errors at the *destination* of these responses.

even parity (event): Determines whether odd or even parity is generated for event data. If this field is *clear*, *odd* parity will be generated. If this field is *set*, *even* parity will be generated. The value of this field lies only in its use as a debugging and commissioning tool to purposely stimulate parity errors at the *destination* of the event.

data masking: Determines whether event data emitted from the AEM's FREE boards is either masked or accepted. Each bit offset corresponds to the masking for a particular FREE. The LSB corresponds to $FREE_0$ and the MSB corresponds to $FREE_{11}$. If a bit is *clear*, any data from the specified FREE is accepted. If the bit is *set*, data is masked.

event: Specifies the time (in units of `sysclk`), while waiting for event data from a cable before declaring a timeout error.

1.3.2 Common status

In general, this register reflects the latched values of the signals the AEM uses in building events. If a signal is asserted, it is latched and the corresponding field is *set*. Note: This register may be either read or written. However, the value specified on write is ignored and the register is always cleared (all fields set to *zero*).

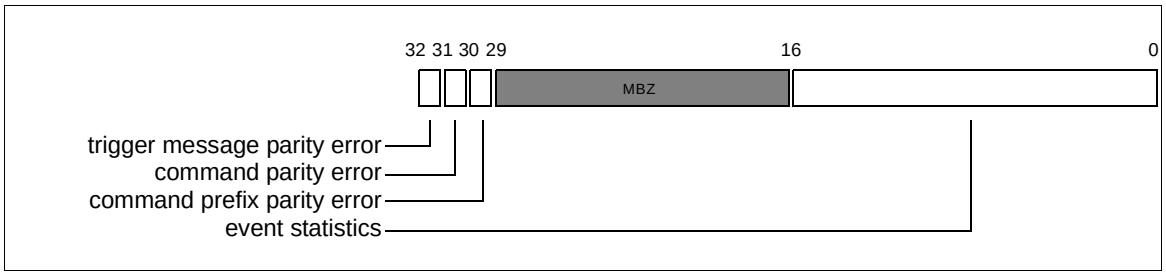

Figure 3 AEM Status register

trigger message parity error: Specifies whether the AEM has ever detected a parity error in an incoming trigger message. If the bit is *set*, a parity error has been detected.

command parity error: Specifies whether the AEM has ever detected a parity error when attempting to decode and process a command directed to *itself*. Parity is evaluated in at least one and possibly two places: the local ***access descriptor***, and in the case of a ***load*** command, the parity associated with the payload. This field is *set* if either of these two evaluations fail.

command prefix parity error: Specifies whether the controller has ever detected a parity error when attempting to decode and process a command's prefix. If the field is *set*, a parity error has been detected in the prefix.

event statistics: The event transmission statistics of the TEM. For example, this field specifies the number of events transmitted by the TEM. As events are sent as LATp packets, the structure of this field corresponds to the ***Transmitter statistics register*** described in [1].

1.3.3 Cable status

This register reflects the latched errors from each of the 12 FREE boards managed and serviced by the AEM. Each board can contribute up to two errors. These error signals are captured in the structure illustrated in Figure 4. If a signal is asserted, it is latched and the corresponding field is *set*. Note: This register may be either read or written. However, the value specified on write is ignored and the register is always cleared (all fields set to *zero*).

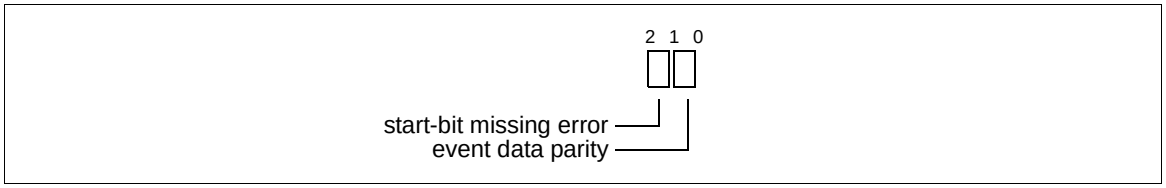

Figure 4 AEM Status register

start-bit missing: Specifies whether the AEM did not detect a start bit before the timeout period. If the bit is *set* a timeout on the start-bit has been detected.

event data parity error: Specifies whether the AEM has ever detected a parity error when attempting to process its input event data. This field is *set* if a parity error occurs.

This register contains the latched error status for the 12 FREE boards. The register is organized as a 2 *x* 12 bit array, where a particular offset of the array corresponds to the errors for the corresponding FREE board as illustrated in Figure 5.

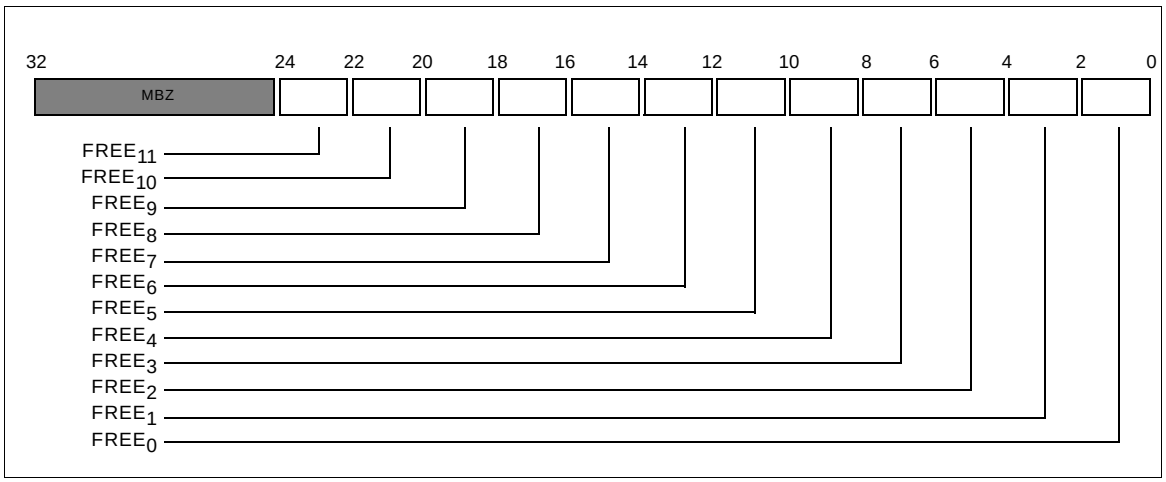

Figure 5 AEM cable status register

1.3.4 Command/Response Statistics register

Note: This register may be either read or written. However, the value specified on write is ignored and the register is always cleared (all fields set to *zero*).

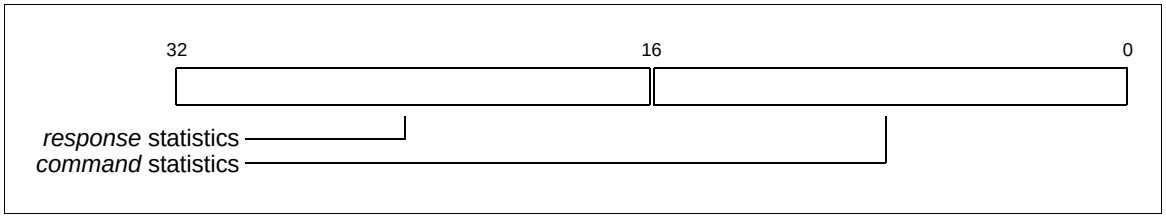

Figure 6 AEM Command/Response statistics register

response statistics: The packet statistics for the outgoing response wire. See [1] for a description of the structure of this field

command statistics: The packet statistics for the incoming command wire. See [1] for a description of the structure of this field

1.3.5 Trigger sequencing register

This register determines the respective timing between the arrival of a ***trigger message*** at the AEM and the subsequent command (or commands) it generates to its Front-End electronics. The two fields of this register are illustrated in Figure 7.

Note: The AEM adds a 5 clock delay between the arrival of the trigger message at the AEM and the issuance of any command.

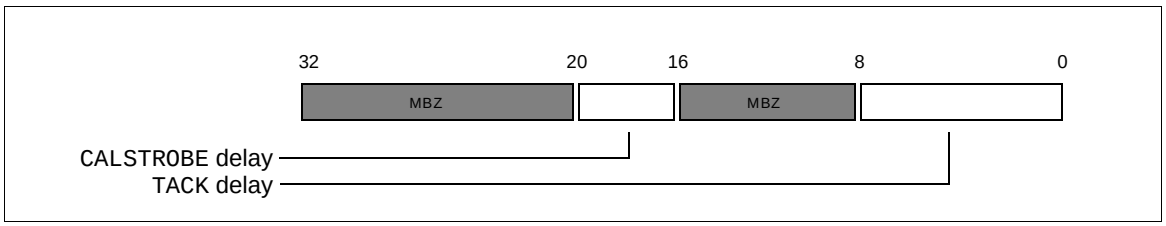

Figure 7 AEM trigger sequencing register

CALSTROBE delay: If the CALSTROBE field of a trigger message is true (*set*), this field specifies the transmission delay (in units of sysclk) of the CCALSTROBE command from the AEM to its Front-End Electronics. The delay is calculated with respect to the arrival time of the trigger message at the AEM. As this is a four-bit field, the maximum delay is 800 nanoseconds. This field is ignored if the CALSTROBE field of the trigger message is false (*clear*). Typically, if a subsystem is operating in isolation, the value of this field is irrelevant. However, if more then one subsystem is operating simultaneously, this field may be used to align the relative starting point of CALSTROBEs from subsystem to subsystem and thus may be used to understand correlated effects between subsystems.

TACK delay: If either the CALSTROBE field of the trigger message is false (*clear*), or the TACK field of a trigger message is true (*set*), this field specifies the transmission delay (in units of sysclk) of the TACK command from the AEM to its FREEs. The delay's starting point depends on the state of the CALSTROBE field of the trigger

message. If the CALSTROBE field is true (*set*), the delay is calculated with respect to the *start* of the transmission of the CALSTROBE. If the CALSTROBE field is false (*clear*), the delay is calculated with respect to the arrival time of the trigger message at the AEM. As this is an eight-bit field, the maximum delay is 12.750 micro-seconds.This field is ignored if the CALSTROBE field of the trigger message is true (*set*) and the TACK field is false *(clear*).

Note: This register may be either read or written. However, the value specified on write is ignored and the register is always cleared (all fields set to *zero*).

1.3.6 Power management

This register is used to turn on and off the power supplied by the AEM to its FREE boards. The structure of this register is illustrated in Figure 8. Each field corresponds to the control of one FREE card. When reading, if a field is *set*, the corresponding power supply is *on*, if the field is *clear*, the supply is *off*. To turn on the power supplies for a particular FREE board, its corresponding field should be *set*. To turn off the power supply, its field should be *cleared*.

Note: on "power-on reset", this register will have a value of zero. In addition, an AEM reset will not affect the state of this register.

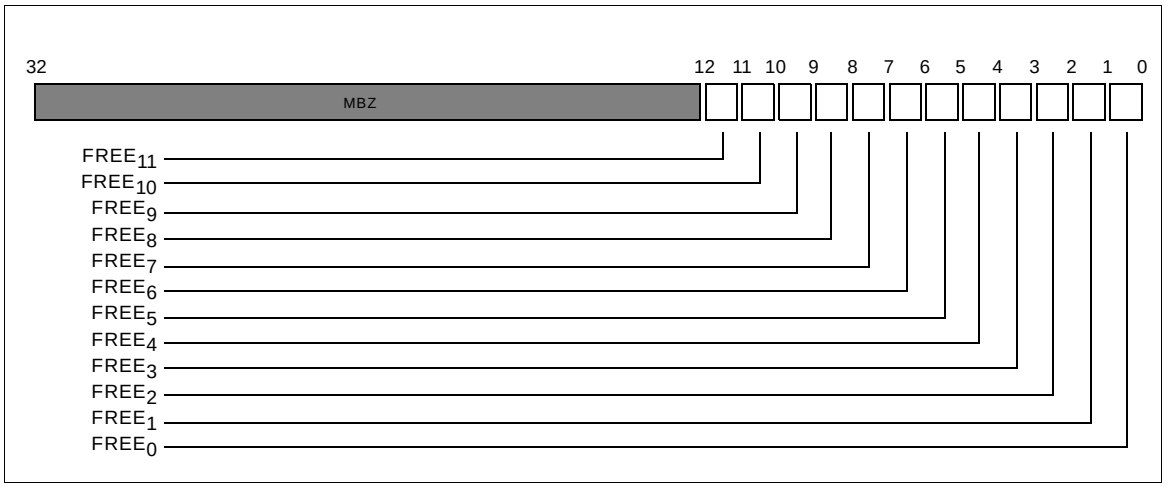

Figure 8 AEM power management

1.4 The Environmental monitor

The AEM monitors various analog quantities associated with its FREE boards. These quantities are digitized and saved in I/O registers contained in the environmental functional block of the AEM. All registers of this block are all sixty-four bits wide. The structure of each of these registers is identical and is represented in Figure 9. Each register represents the analog

quantities monitored for each FREE board. In reality each register is “fronting” for four individual ADCs. Each of the four ADCs are identical (a commercial part, the MAXIM 145, see [4]). Four 12 bit fields of this register reflect the converted four values of the ADCs. The correspondence between register number and FREE board is enumerated in Table 5. To begin conversion, the user *writes* the register (the value written is ignored). This will force the ***not ready*** field to be *set*. At a fixed time later, conversion is complete and the ***not ready*** field will be *clear* and the four ADC value fields will have legitimate values. If the register is written while a conversion is in progress, the AEM will ignore the write.

Table 5 The environmental monitor *register*s

Name	Number	Access	Description
ENV_FREE_00	0	R/W	Environmental quantities for $FREE_0$
ENV_FREE_01	1	R/W	Environmental quantities for $FREE_1$
ENV_FREE_02	2	R/W	Environmental quantities for $FREE_2$
ENV_FREE_03	3	R/W	Environmental quantities for $FREE_3$
ENV_FREE_04	4	R/W	Environmental quantities for $FREE_4$
ENV_FREE_05	5	R/W	Environmental quantities for $FREE_5$
ENV_FREE_06	6	R/W	Environmental quantities for $FREE_6$
ENV_FREE_07	7	R/W	Environmental quantities for $FREE_7$
ENV_FREE_08	8	R/W	Environmental quantities for $FREE_8$
ENV_FREE_09	9	R/W	Environmental quantities for $FREE_9$
ENV_FREE_10	10	R/W	Environmental quantities for $FREE_{10}$
ENV_FREE_11	11	R/W	Environmental quantities for $FREE_{11}$
Total	**12**		

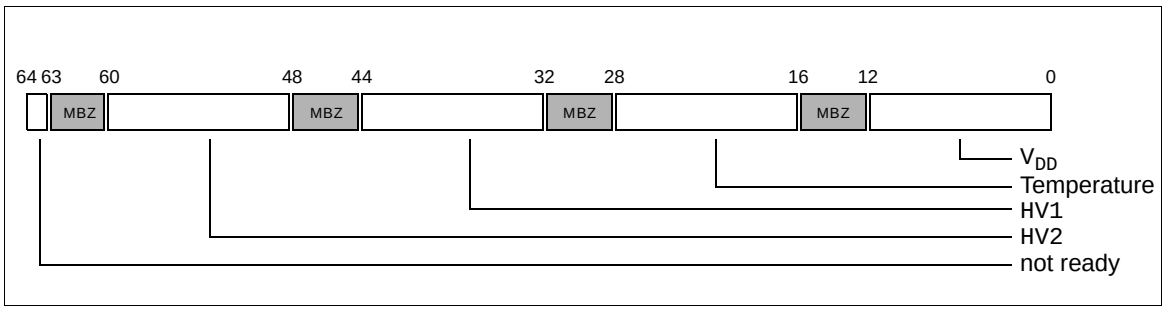

Figure 9 Monitoring register

1.5 The GLAST ACD Readout Control (GARC)

Table 6 The GARC registers (function block 0)

Name	Number	Access	Reg #	Description
VETO_DELAY	2	R / W	2	Delay from DISC in to NVETO out
RQST_HVBS	8	R / W	8	Requested High Voltage (HV) for normal operation
RQST_HVSAA	9	R / W	9	Requested High Voltage (HV) for SSA operation
HVBS	10	read *only*[1]	10	Current High Voltage (HV) for normal operation
HVSAA	11	read *only*[1]	11	Current High Voltage (HV) for SSA operation
HOLD_DELAY	12	R / W	12	Delay from trigger to hold
VETO_WIDTH	13	R / W	13	Pulse width of NVETO
HITMAP_WIDTH	14	R / W	14	Minimum pulse width of hitmap signals
HITMAP_DEADTIME	15	R / W	15	Time added to hitmap signals
Total	**9**			

1. See Table 14 on page 32

Table 7 The GARC registers (function block 1)

Name	Number	Access	Reg #	Description
LOOK_AT_ME	20	R / W	4	A or B only
HITMAP_DELAY	24	R / W	8	Delay from DISC in to HITMAP signals
PHA_EN_0	25	R / W	9	PHA readout enables for channels $_{0\text{-}15}$
VETO_EN_0	26	R / W	10	Veto enables for channels $_{0\text{ -}15}$
HLD_EN_0	27	R / W	11	HLD enables for channels $_{0\text{ -}15}$
PHA_EN__1	28	R / W	12	PHA readout enables for channels $_{16\text{ -}17}$ (in LSB)
VETO_EN__1	29	R / W	13	Veto enables for channels $_{16\text{ -}17}$ (in LSB)
HLD_EN__1	30	R / W	14	HLD enables for channels $_{16\text{ -}17}$ (in LSB)
MAX_PHA	31	R / W	15	Maximum allowable number of PHA values
Total	**9**			

Table 8 The GARC registers (function block 2)

Name	Number	Access	Reg #	Description
MODE	40	R / W	8	Various bit- fields for mode settings
STATUS	41	read only	9	Status
LAST_CMND	42	read only	10	Command or data from last command error
DIAGNOSTIC	43	read only	11	?
CMD_REJECT	44	read only	12	Number of commands rejected
FREE_ID	45	read only	13	FREE board ID
GARC_VERSION	46	read only	14	GARC version number
Total	**7**			

Table 9 The GARC registers (function block 3)

Name	Number	Access	Reg #	Description
PHA_THRESHOLD_0	56	R / W	8	PHA threshold for channel $_0$
PHA_THRESHOLD_1	57	R / W	9	PHA threshold for channel $_1$
PHA_THRESHOLD_2	58	R / W	10	PHA threshold for channel $_2$
PHA_THRESHOLD_3	59	R / W	11	PHA threshold for channel $_3$
PHA_THRESHOLD_4	60	R / W	12	PHA threshold for channel $_4$
PHA_THRESHOLD_5	61	R / W	13	PHA threshold for channel $_5$
PHA_THRESHOLD_6	62	R / W	14	PHA threshold for channel $_6$
Total	**7**			

Table 10 The GARC registers (function block 4)

Name	Number	Access	Reg #	Description
PHA_THRESHOLD_7	72	R / W	8	PHA threshold for channel $_7$
PHA_THRESHOLD_8	73	R / W	9	PHA threshold for channel $_8$
PHA_THRESHOLD_9	74	R / W	10	PHA threshold for channel $_9$
Total	**7**			

Table 10 The GARC registers (function block 4)

Name	Number	Access	Reg #	Description
PHA_THRESHOLD_10	75	R/W	11	PHA threshold for channel $_{10}$
PHA_THRESHOLD_11	76	R/W	12	PHA threshold for channel $_{11}$
PHA_THRESHOLD_12	77	R/W	13	PHA threshold for channel $_{12}$
PHA_THRESHOLD_13	78	R/W	14	PHA threshold for channel $_{13}$
Total	**7**			

Table 11 The GARC registers (function block 5)

Name	Number	Access	Reg #	Description
PHA_THRESHOLD_14	88	R/W	8	PHA threshold for channel $_{14}$
PHA_THRESHOLD_15	89	R/W	9	PHA threshold for channel $_{15}$
PHA_THRESHOLD_16	90	R/W	10	PHA threshold for channel $_{16}$
PHA_THRESHOLD_17	91	R/W	11	PHA threshold for channel $_{17}$
ADC_TACQ	92	R/W	12	ADC acquisition time
Total	**5**			

1.6 The GLAST ACD Front-End Controller (GAFE)

Table 12 The GAFE registers

Name	Number	Access	Description
CONFIGURATION	0	R/W	
VETO_DAC	1	R/W	
HLD_DAC	2	R/W	
LLD_DAC	3	R/W	
BIAS_DAC	4	R/W	Bias value
TCI_DAC	5	R/W	Test charge inject value
VERS_ADDR	6	read *only*	Version number
Total	**11**		

Table 12 The GAFE registers

Name	Number	Access	Description
WRITE_CTR	7	read *only*	Number of load commands since reset
REJECT_CTR	8	read *only*	Number of commands rejected since reset
LOOP_CTR	9	read *only*	Number of commands since reset
CHIP_ADDR	10	read *only*	GAFE chip address
Total	**11**		

Chapter 2 Commanding

2.1 Overview

2.1.1 Conventions

All data structures described in this chapter are from the perspective of being “on-the-wire”. Therefore, the left-most field in any description is transmitted *first*, or is considered to be transmitted on the *zeroth* clock. Fields are numbered from the beginning of the ***command string*** described in [1].

2.1.2 Introduction

This chapter describes the remote protocol necessary to access both the registers[1] and functional blocks of the AEM. It draws on the Command/Response Protocol discussed in [1]. Commanding is hierarchical, i. e., to access any specific register or functional block it is necessary to specify a hierarchical path to that block or register. Thus, for example, to access a register in an ACD Front-End ASIC (GAFE), one must go through a AEM and GARC. The commanding hierarchy of the AEM is illustrated in Figure 10. Note that operations are defined on both the registers of a functional block and on the block *itself*. Access to the functions of block are commonly known as a performing a ***dataless command***.

1. Enumerated and described in Chapter 1

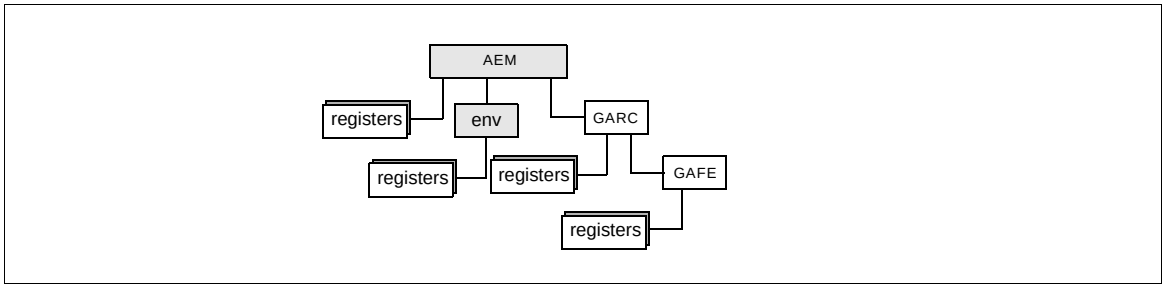

Figure 10 Hierarchy of target types

Two of the AEM's functional blocks are located physically on the AEM as specified in the shaded blocks of Figure 10. Access to on-board or off-board blocks is specified in the first 9-bits, or *prefix* of the ***command string***. As there are two on-board blocks and one off-board block, there are three possibilities for the prefix as shown in Figures 11, 12 and 13.

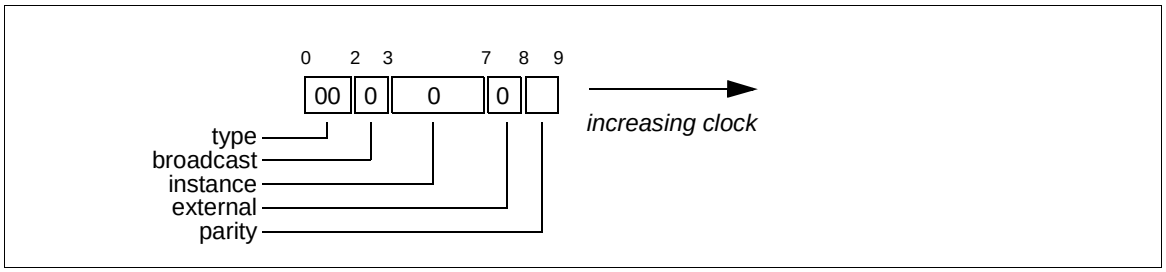

Figure 11 Command string prefix for accessing the Common Controller of the AEM

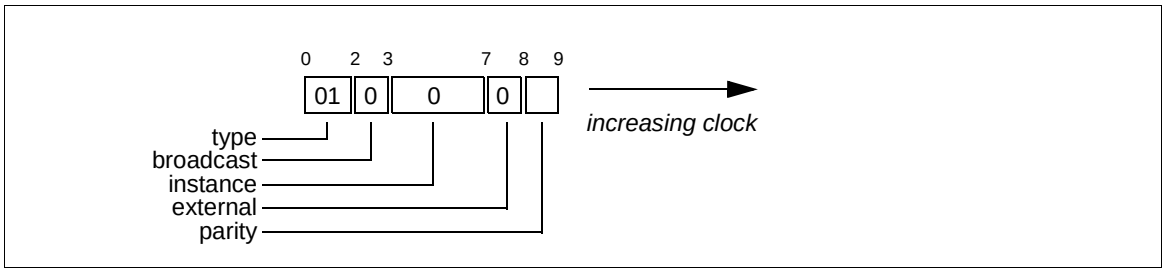

Figure 12 Command string prefix for accessing the Environmental Monitor of the AEM

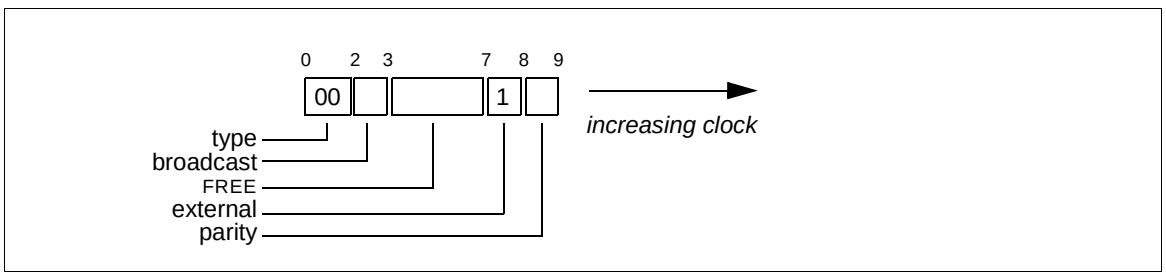

Figure 13 Command string prefix for accessing off-board functional blocks and registers of the AEM

2.2 The AEM's access descriptors

The AEM has a two on-board functional blocks; the first called the Common Controller and the second called the environmental monitor. Each of these block types have their own set of internal registers. In addition the AEM services 12 FREE boards, each of which has its own functional blocks and registers. These FREE boards constitute the AEM's *off*-board functional blocks and registers. Thus, the AEM has *two* different kinds of ***access descriptors***:

ACD: The command will be relayed by the AEM to its appropriate off-board block (on a FREE board) where the command will actually be executed.

Local: The access is constrained entirely within the Common Controller.

2.2.1 ACD access descriptor

The ACD access descriptor allows access to all off-board functional blocks and registers. The access descriptor when used as part of a command is prefixed with fixed four-bit start pattern as illustrated in Figure 14. The access descriptor is also returned as one part of a response to a Read command. However, in this case it does *not* include the start pattern (see, for example Section 2.6.3).

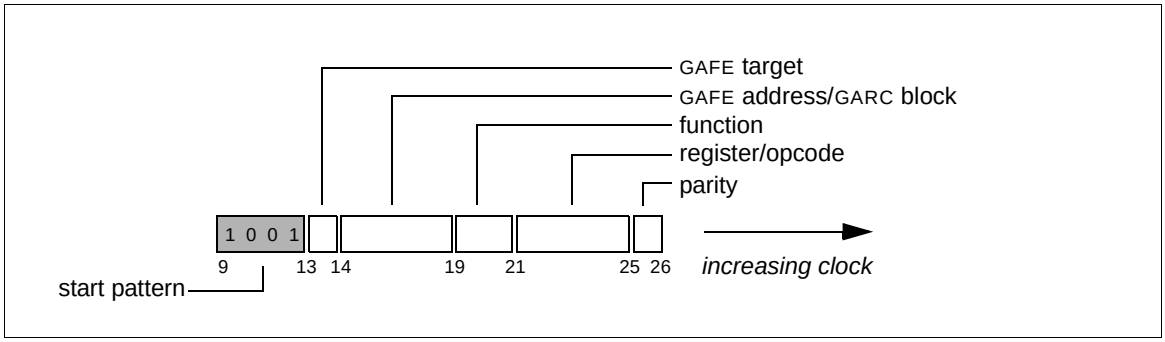

Figure 14 Access descriptor for external ACD commands

GAFE target: This field determines whether the command is intended for the GARC itself, or should be forwarded to one or more of the GAFEs managed by the GARC. If the field is *true*, the command is forwarded to the GAFE whose address is determined by the ***GAFE address/GARC block*** field and the interpretation of the ***register/opcode*** field is GAFE specific. If the bit is *false*, the command is forwarded to the functional block determined by the ***GAFE address/GARC block*** field and the interpretation of the ***register/opcode*** field is GARC specific.

GAFE address/GARC block: The address of the functional block targeted by this command. Its interpretation is dependent on the ***GAFE target*** field.

— If the ***GAFE target*** field is *true*, this field corresponds to the address of a GAFE. Their are 18 possible GAFE addresses varying from zero (0) to seventeen (17). In addition, a value of `0x1F` (all bits *set*) targets *all* the GAFEs managed by the specified GAFE. This address is called the *broadcast* address. The broadcast address is not permitted if the ***function*** field specifies a `read` operation.

— If the ***GAFE target*** field is *false*, this field corresponds to a function block within the specified GARC. Function blocks are number from zero (0) through five (5). Within each function block are a number of registers. A specific register within the function block register is determined by the value of the ***register/opcode*** field.

function: Enumerates what *type* of access is required of the target, by the command[1]. For example, whether the command will either *read* or *write* the specified register. The valid enumerations for this field are described in [1].

register/opcode: If the ***function*** field has the value `read` or `load`, this field contains the *number* of the register to be accessed. If the function is `dataless`, this field determines the *type* of dataless access. The interpretation of this field depends also on the value of the ***GAFE target*** field.

parity: The *odd* parity value over the entire ***access descriptor*** *less* the starting fields (these are the greyed-out fields).

2.2.2 Local Access Descriptor

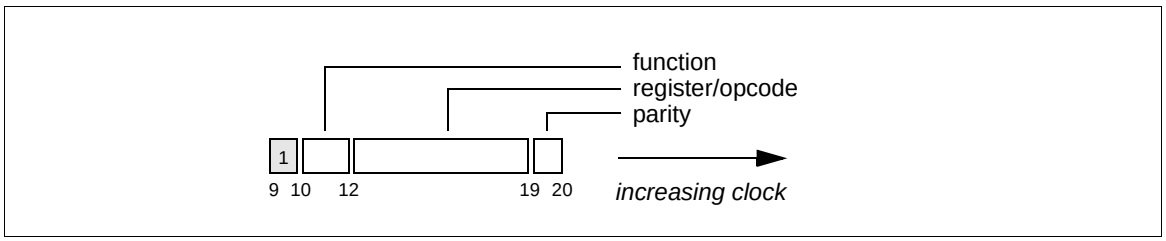

Figure 15 Access descriptor for local commands

function: Enumerates what *type* of access is required of the target, by the command. For example, whether the command will either *read* or *write* the specified register. The valid enumerations for this field are described in [1].

register/opcode: If the ***function*** field has the value `read` or `load`, this field contains the *number* of the register to be accessed. If the function is `dataless`, this field determines the *type* of dataless access.

parity: The *odd* parity value over the entire ***access descriptor*** *less* the first (or start) bit.

1. ACD doesn't really support dataless functions. However, as designed it *does* ignore the dataless function code and should treat it as a write function. It must continue to do in order to meet LAT communications standards.

2.3 Accessing the Common Controller (or AEM)

2.3.0.1 Dataless commands

Table 13 The Common Controller's dataless commands

Name	Opcode	Description
Reset	1	Hard reset of the AEM
Total	**1**	

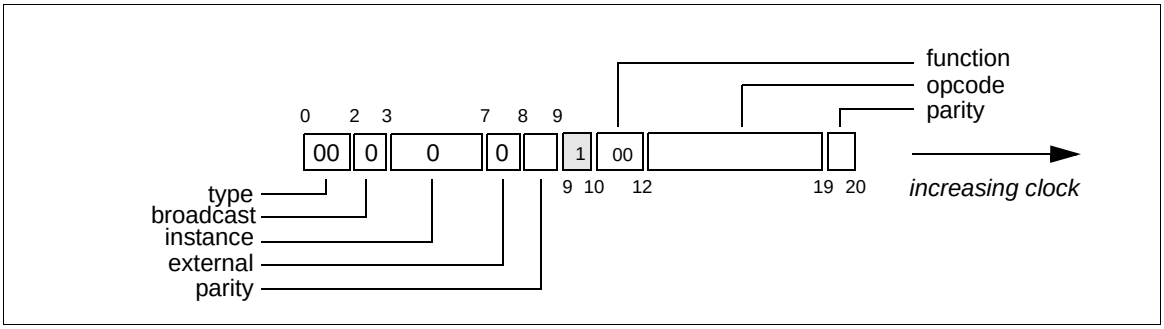

Figure 16 Access descriptor for AEM dataless functions

2.3.0.2 Load commands

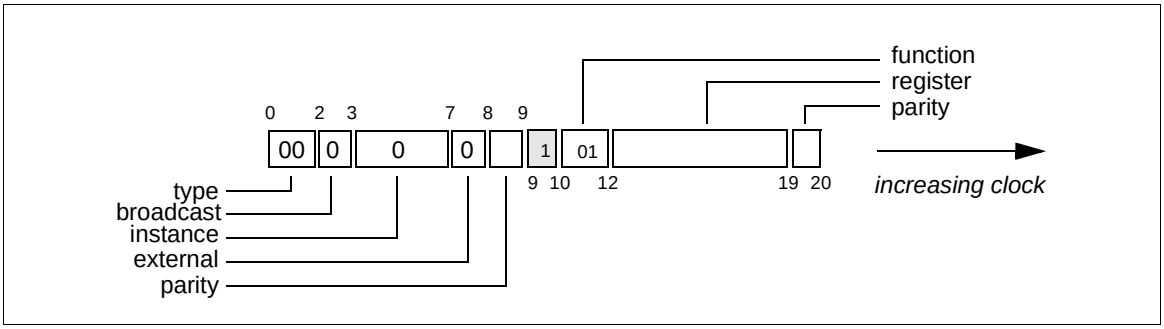

Figure 17 Access descriptor for AEM register load commands

Load functions require a 33-bit payload, as they are protected with a trailing parity field. The computed value of the parity field is the *odd* parity over the preceding 32 bits of data which are to be loaded into the register. The format of this payload is illustrated in Figure 18. As a load requires no response, the ***Respond*** field of the packet is set to *false*.

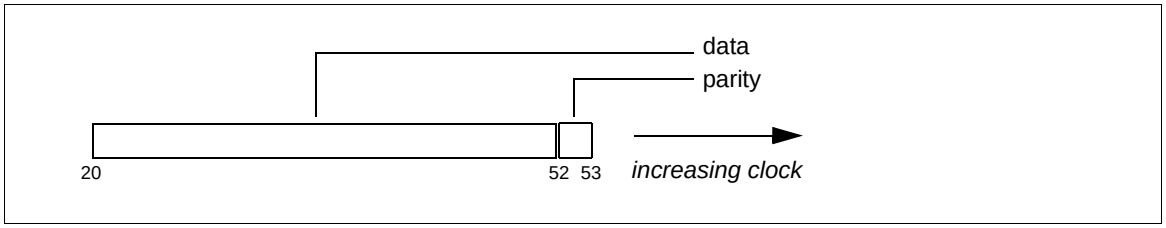

Figure 18 Payload for AEM register load commands

2.3.0.3 Read commands

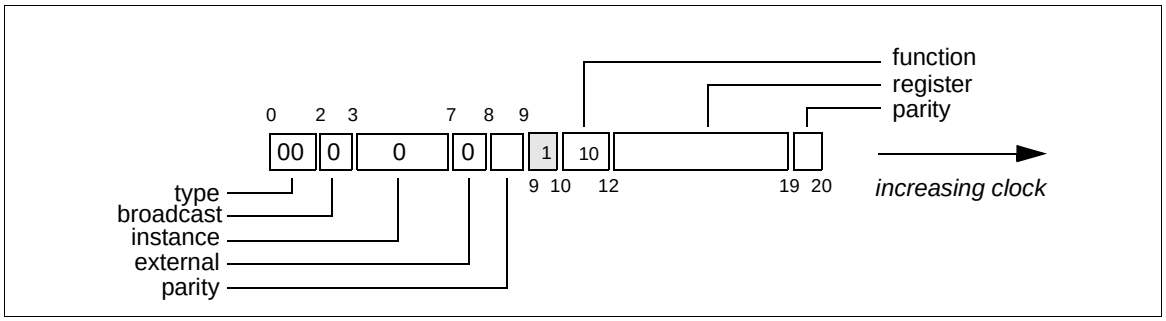

Figure 19 Access descriptor for AEM register read commands

`Read` functions require *no* payload. The value of the register read is returned as a response. As these `reads` *do* generate a response, the command packet's ***Respond*** field is set to *true*. The format of that response is illustrated in Figure 20.

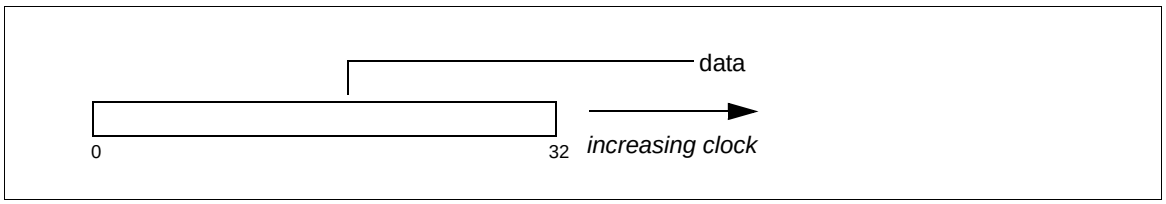

Figure 20 Response to AEM register read commands

2.4 Accessing the environmental monitor

2.4.0.1 Load commands

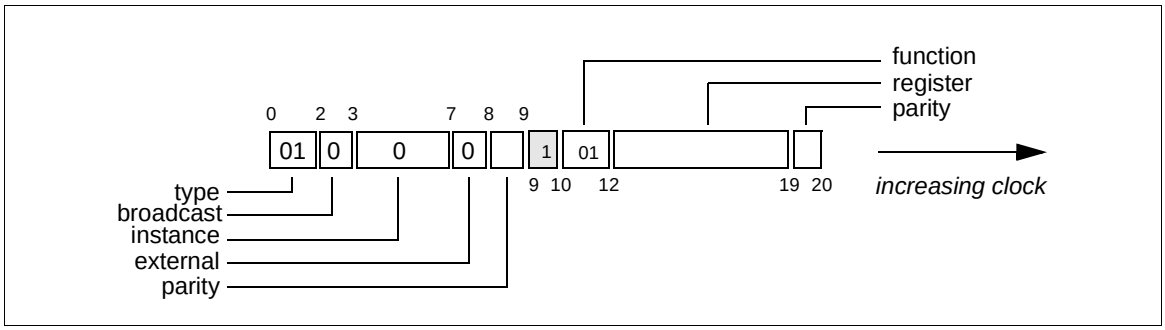

Figure 21 Access descriptor for environmental monitor register load commands

`Load` functions require a 65-bit payload, as they are protected with a trailing parity field. The computed value of the parity field is the *odd* parity over the preceding 64 bits of data which are to be loaded into the register. The format of this payload is illustrated in Figure 22. As a `load` requires no response, the ***Respond*** field of the packet is set to *false*.

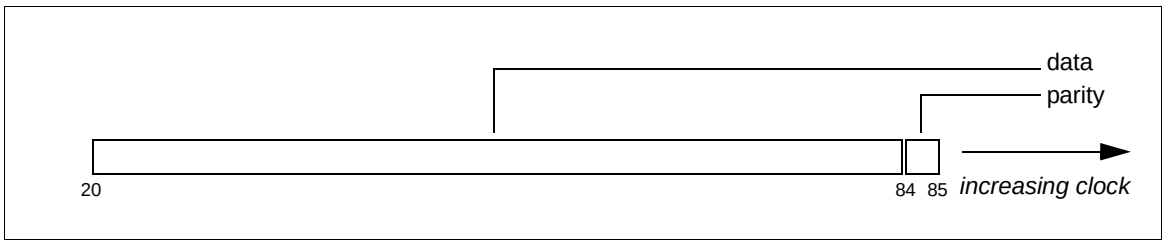

Figure 22 Payload for environmental monitor register load commands

2.4.0.2 Read commands

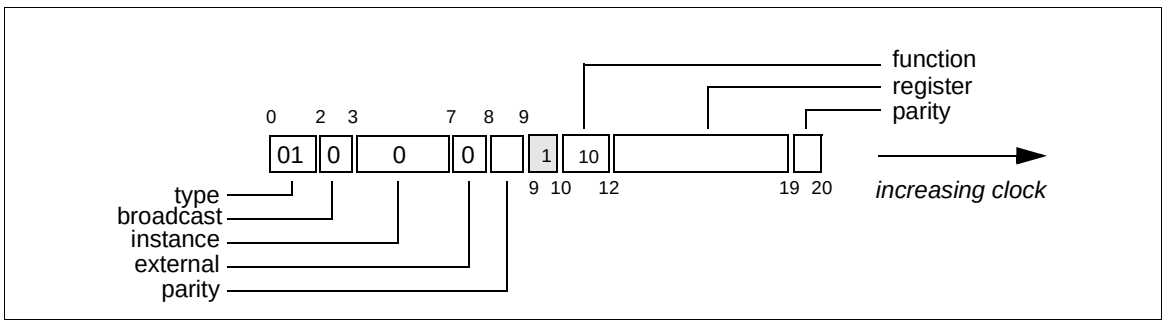

Figure 23 Access descriptor for environmental monitor register read commands

`Read` functions require *no* payload. The value of the register read is returned as a response. As these `reads` *do* generate a response, the command packet's ***Respond*** field is set to *true*. The format of that response is illustrated in Figure 24.

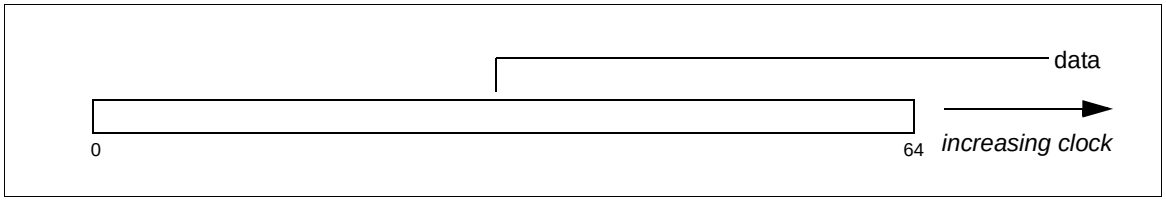

Figure 24 Response to environmental monitor register read commands

2.5 The GLAST ACD Readout Controller (GARC)

2.5.1 Dataless commands

Table 14 The GARC dataless commands

Name	Opcode	Function Block	Register	Description
RESET	1	0	1	Hard reset of the GARC
CALSTROBE[1]	3	0	3	Generate calibration strobe
SET_HVBS[2]	10	0	10	Set HV to normal value
SET_HVSSA[2]	11	0	11	Set HV to SSA value
Total	**4**			

1. Reserved for internal use to AEM
2. See Table 6 on page 21

`Dataless` functions *do* require a payload. The structure of the payload is the same as the 17-bit payload for `Load` functions, however, the data portion must be zero. The format of this payload is illustrated in Figure 26. As `dataless` functions require no response, the ***Respond*** field of the packet is set to *false*.

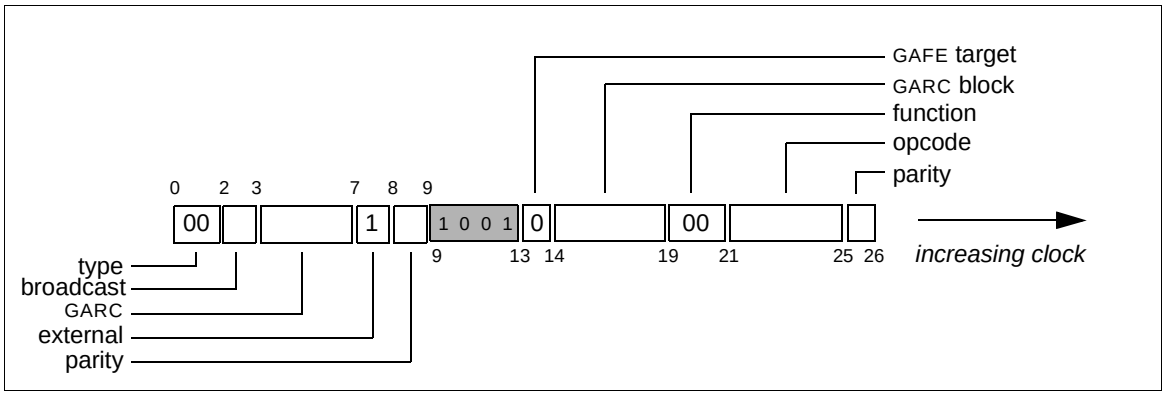

Figure 25 Access descriptor for GARC dataless commands

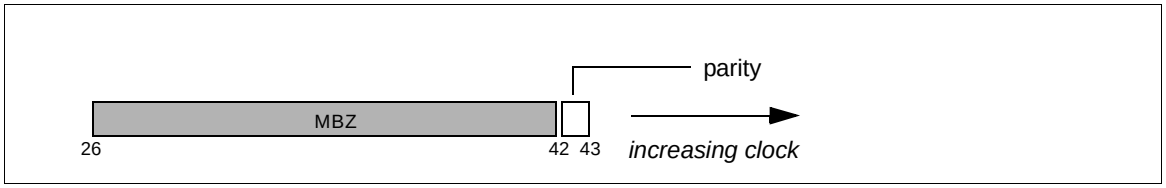

Figure 26 Payload for GARC dataless and read commands

2.5.2 Load commands

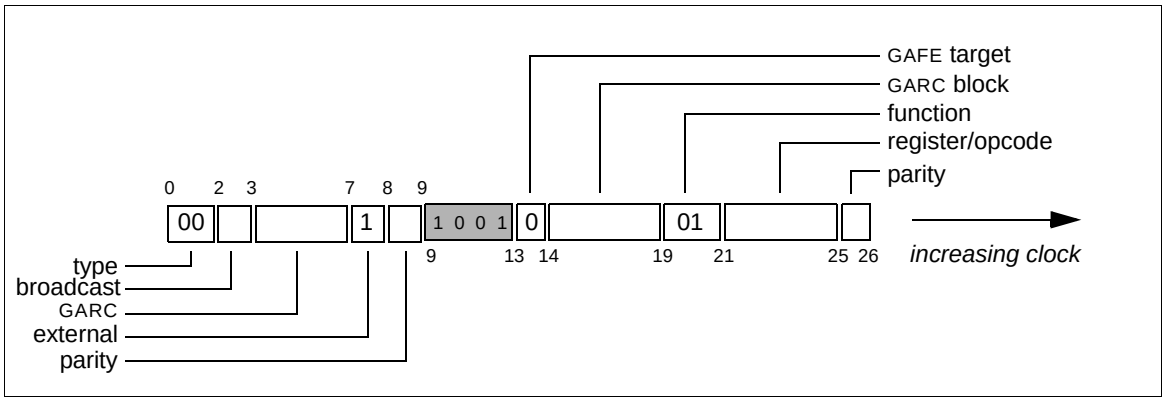

Figure 27 Access descriptor for GARC register load commands

Load functions require a 17-bit payload. The format of this payload is illustrated in Figure 28. As a load requires no response, the ***Respond*** field of the packet is set to *false*.

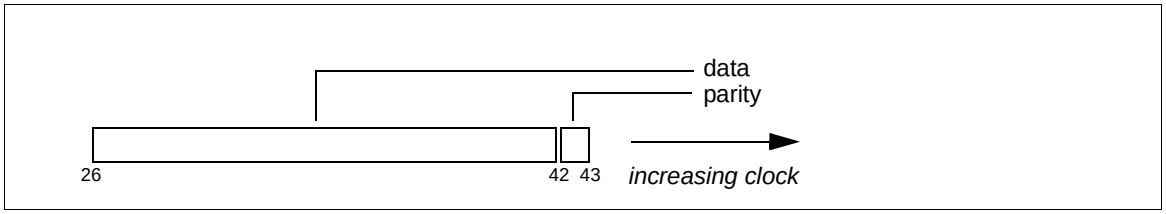

Figure 28 Payload for GARC register load commands

2.5.3 Read commands

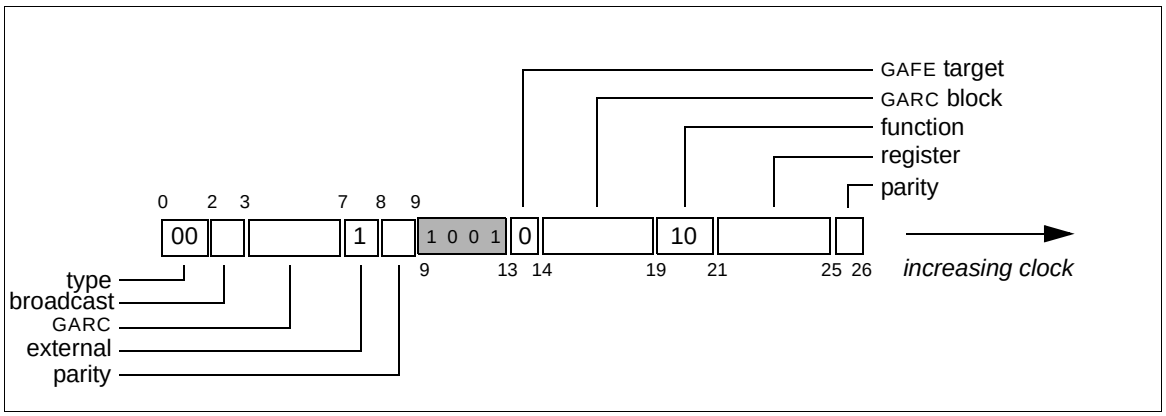

Figure 29 Access descriptor for GARC register read commands

`Read` functions *do* require a payload. The structure of the payload is the same as the 17-bit payload for `Load` functions, however, the data portion must be zero. The format of this payload is illustrated in Figure 26. The value of the register read is returned as a response. As these `reads` *do* generate a response, the command packet's ***Respond*** field is set to *true*. The format of that response is illustrated in Figure 30.

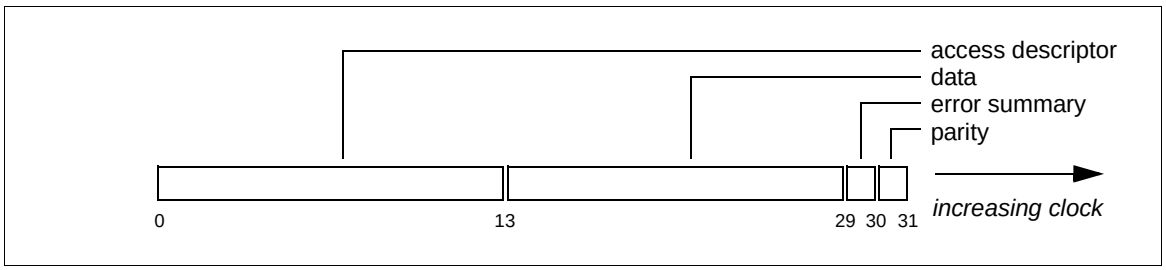

Figure 30 ACD response to GARC register read commands

Note, that the returned access descriptor does *not* include the start pattern (see Section 2.2.1).

2.6 The GLAST ACD Front-End Controller (GAFE)

2.6.1 Dataless commands

none...

2.6.2 Load commands

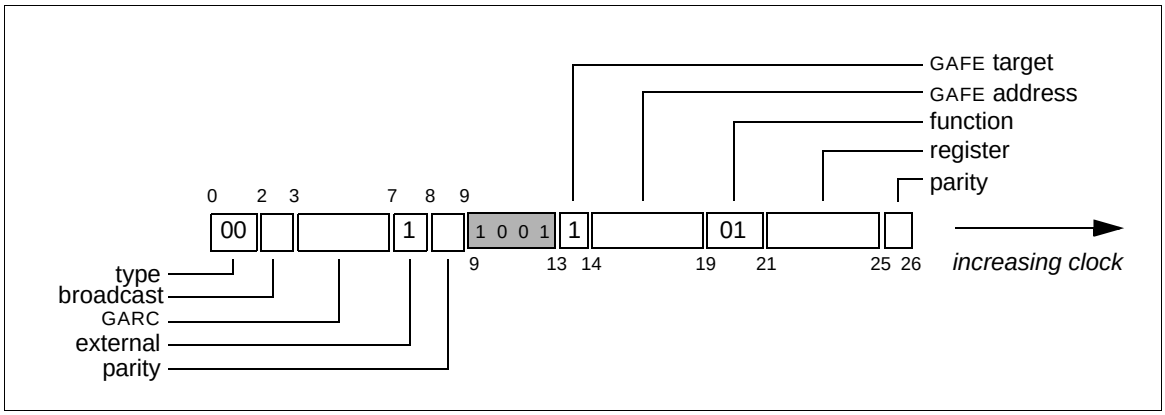

Figure 31 Access descriptor for GAFE register load commands

Load functions require a 17-bit payload. The format of this payload is illustrated in Figure 32. As a load requires no response, the ***Respond*** field of the packet is set to *false*.

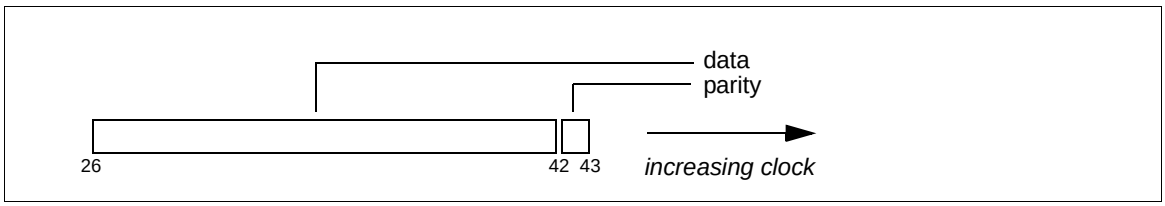

Figure 32 Payload for GAFE register load commands

2.6.3 Read commands

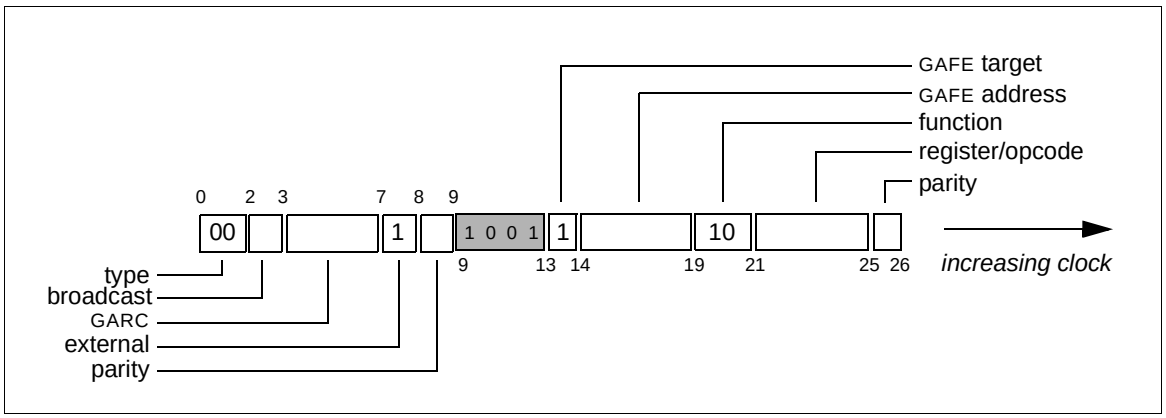

Figure 33 Access descriptor for GAFE register read commands

`Read` functions *do* require a payload. The structure of the payload is the same as the 17-bit payload for `Load` functions, however, the data portion must be zero. The format of this payload is illustrated in Figure 26. The value of the register read is returned as a response. As these `reads` *do* generate a response, the command packet's ***Respond*** field is set to *true*. The format of that response is illustrated in Figure 34.

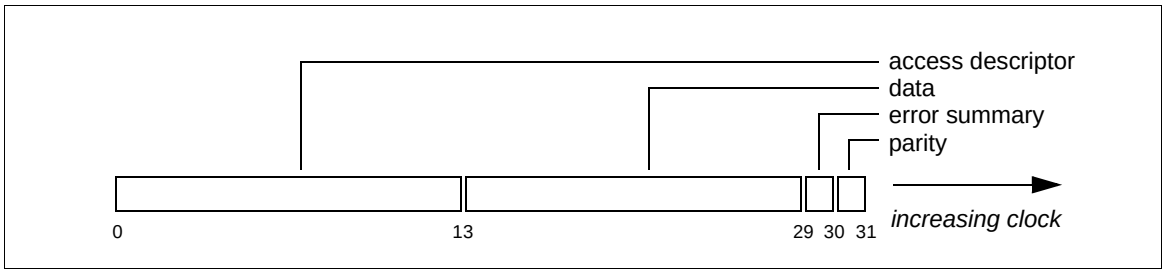

Figure 34 Response to GAFE register read commands

Note, that the returned access descriptor does *not* include the start pattern (see Section 2.2.1).

Chapter 3 Events

3.1 Introduction

This chapter describes the format of the event data generated by the ACD electronics of the AEM in response to receiving a trigger message. The *shape* of the AEM's event data is governed by the Event Data Protocol (EDP) described in [1].

3.1.1 Background

The ACD consists primarily of scintillating tiles[1]. The light output of any one tile is captured *twice* by two individual fibres, in order to satisfy redundancy requirements. Currently, both fibres are envisioned as being always active, with a fibre's data being ignored when it fails. Consequently, while from the perspective of fail-over each fibre stands alone, it is convenient from a structural perspective to consider tiles as serviced by a fibre *pair*. One half of the pair is referred to as the *a-fibre* and the other half as the *b-fibre*. The entire ACD contains a total of 216 fibres, or 108[2] fibre pairs. The servicing of one fibre is done through an ASIC called the GLAST ACD Front-End (GAFE). In turn, 18 GAFEs are managed and serviced by another ASIC called the GLAST ACD Read-out Controller (GARC). A GARC and its GAFEs are all contained on a single board called the FREE. Data is brought from each FREE board to the AEM on a single cable, consequently it is convenient to think of the AEM as servicing *twelve* cables; 6 for the *a* fibres and 6 for the *b* fibres.

3.2 The event contribution

From the perspective of event read-out it is convenient to consider the ACD as emitting data from twelve *cables*, with the data from each cable each cable producing information from eighteen individual *channels*. A channel corresponds to the information from a single fibre

1. I ignore the so-called "duct tape".
2. Actually, potential electronics channels. Geometric considerations leave some channels unoccupied.

serviced through one GAFE. A cable corresponds to the data aggregate of eighteen GAFEs serviced through one GARC. Each channel of the ACD is cable of producing three kinds of information:

- Discriminated *hit* (or *veto*) information
- Discriminated *accept* information
- Zero suppressed, *pulse height* information

Thus, each cable emits both a fixed and variable sized component. The total contribution of the AEM is just the sum of the twelve cables it services, as illustrated in Figure 35:

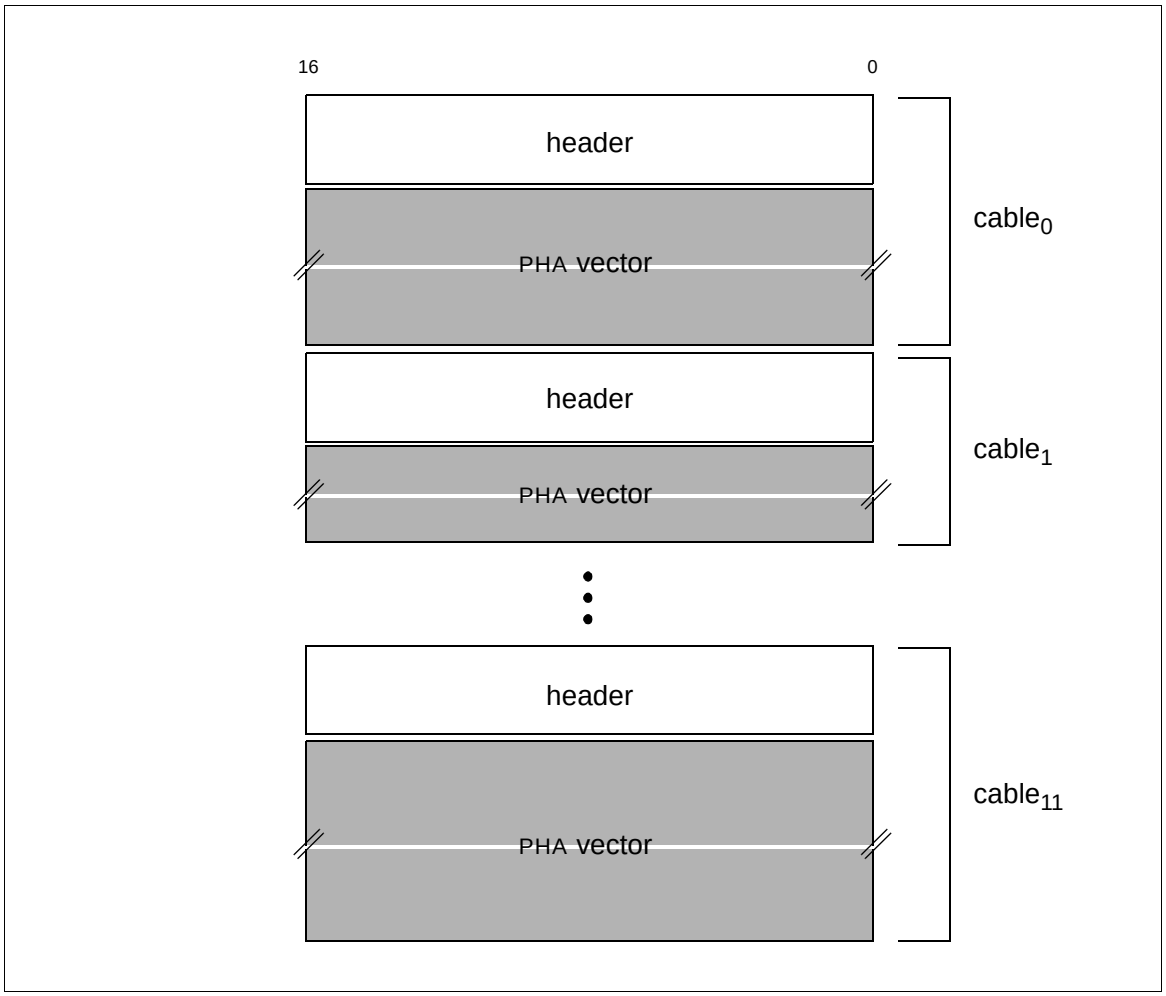

Figure 35 Overall structure of the AEM's natural event data

3.2.1 The cable header

Each contribution from each cable begins with a fixed 48 bits of information as illustrated in Figure 36:

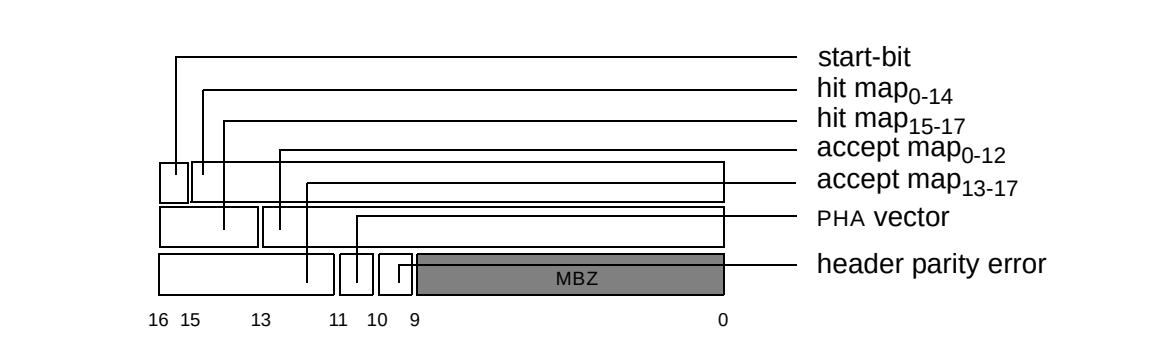

Figure 36 Structure of cable header

start-bit: Specifies whether the AEM detected a start bit before the timeout period. If the bit is *set* a the start-bit for the cable was detected. One may infer, that if this field is *clear* a timeout on the cable occurred.

hit map (0-14): The first fifteen channels of the hit map. If a bit at a specified offset is *set*, the corresponding channel is hit. Channels are organized in *ascending* order. The LSB bit offset corresponds to the high-order channel. For example, if bit offset 0 is *set*, a hit occurred on channel 0, or if bit offset 14 is *set*, a hit occurred on channel 14.

hit map (15-17): The last three channels of the hit map. If a bit at a specified offset is *set*, the corresponding channel is hit. Channels are organized in *ascending* order. The LSB bit offset corresponds to the high-order channel. For example, if bit offset 13 is *set*, a hit occurred on channel 15, or if bit offset 15 is *set*, a hit occurred on channel 17.

accept map (0-12): The first thirteen channels of the zero suppression map. If a bit at a specified offset is *set*, the corresponding channel's PHA discriminator was above threshold. Note, this does *not* necessarily imply that a PHA value was emitted (see Section 3.2.3) as part of the event data. Channels are organized in *ascending* order. The LSB bit offset corresponds to the high-order channel. For example, if bit offset 0 is *set*, channel 0 was above threshold.

accept map (13-17): The last five channels of the zero suppression map. If a bit at a specified offset is *set*, the corresponding channel's PHA discriminator was above threshold. Note, this does *not* necessarily imply that a PHA value was emitted (see Section 3.2.3) as part of the event data. Channels are organized in *ascending* order. The LSB bit offset corresponds to the high-order channel. For example, if bit offset 11 is *set*, channel 13 was above threshold, or if bit offset 15 is set, channel 17 was above threshold.

PHA vector: Specifies whether one or more PHA values follow the header. If this field is *set*, at least one PHA value follows the header. If the field is clear, there were no PHA values emitted by the cable for this event.

header parity error: Specifies whether the AEM detected a parity error when attempting to process its input event data. This field is *set* if a parity error occurs.

3.2.2 The PHA vector

Following the ***header*** is a *n x* 16-bit vector containing Pulse Height data. Each element of this vector is a Pulse Height Analysis (PHA) value. The format of a PHA value is illustrated in Figure 37.

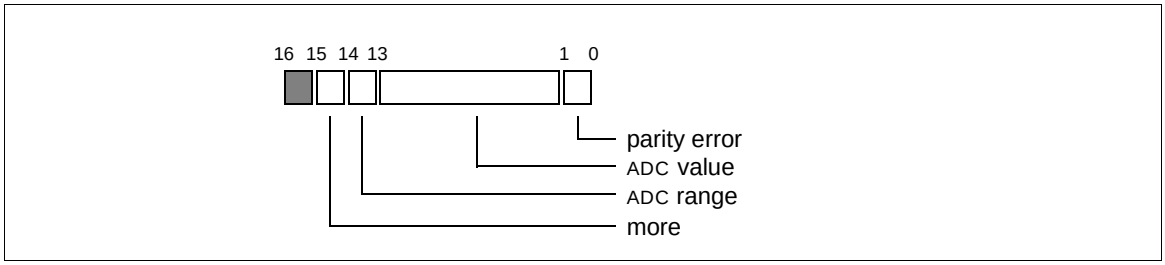

Figure 37 Structure of a PHA value

parity error: Parity is computed on each PHA value sent on a cable. If the parity calculation fails, this field is *set*. Note, that PHA parity errors are *not* included in the event summary.

ADC value: The 12-bit ADC value of the PHA.

ADC range: Specifies one of the two different ADC ranges. The field is *set* for high range, *clear* for low range.

more: Specifies whether additional PHA values follow. If another PHA value follows this field is *set*. If this is the last PHA value this field is *clear*.

Note that the PHA vector is *only* present if the ***PHA vector*** field of the header is *set*. If present, the length of the vector is determined by following the ***more*** field of each PHA value of the vector.

3.2.3 Mapping the accept map to PHA channels

Naively, one might consider the mapping between the PHA values emitted and the ***accept-map*** to be unambiguous. For example, the count of *set* bits in the ***accept-map*** as equal to the length of the PHA vector. However, this is not necessarily the case for three reasons:

i. In order to cap ACD deadtime[1], the maximum number of PHA values accepted by any one cable is *cutoff* at some arbitrary point. This point is determined by the value of the MAX_PHA register in the GARC[2]. The ***accept-map*** represents the number of accepts *without* taking MAX_PHA into account. If the count of set bits is *greater* then MAX_PHA, the contribution was truncated. As PHA values are emitted by a cable in increasing channel order, it will always be the most significant channels which are cutoff.

ii. The ACD electronics allows inhibiting (on a channel by channel basis) the emission of PHA values. The corresponding channels of the ***accept-mask*** do *not* reflect this fact.

1. Which is proportional to the amount of information, or PHA values on the “wire” between ACD and AEM.

iii. A trigger request may inhibit zero suppression (i., e., allow for all 18 channels of a cable to be emitted, independent of whether they are over threshold). In this case, there is no guarantee that *all* channels of the ***accept-mask*** will be set.

For these reasons, the length of the PHA vector should be determined by, first determining whether a vector is present (using the ***PHA vector*** field of the header) and if so, following the ***more*** field of each PHA value. In addition, as the channel mapping is ambiguous without subsidiary information, it is recommended that the MAX_PHA and PHA enable features of the ACD electronics *not* be used.

3.3 The error contribution

There is *no* error contribution.

3.4 The diagnostic contribution

There is *no* diagnostic contribution.

2. See Table 7 on page 21.

Index

M

N

P

R

S

T