The GLAST experiment at SLAC

LAT Inter-module Communications

Electronics group

A reference manual

Document Version:	2.2
Document Issue:	2
Document Edition:	English
Document Status:	Draft - for internal distribution only
Document ID:	LAT-TD-00606
Document Date:	July 14, 2004

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

This document has been prepared using the Software Documentation Layout Templates that have been prepared by the IPT Group (Information, Process and Technology), IT Division, CERN (The European Laboratory for Particle Physics). For more information, go to http://framemaker.cern.ch/.

Abstract

To be written.

Hardware compatibility

This document assumes the following hardware revisions: TBD.

Intended audience

This document is intended principally as a guide for the *users* of the modules of the LAT. These include:

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

Conventions used in this document

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

- Field names are shown in bold and italics (*e.g.*, ***respond*** or ***parity***).
- Acronyms are shown in small caps (*e.g.*, SLAC or TEM).
- Hardware signal or register names are shown in Courier bold (*e.g.*, `RIGHT_FIRST` or `LAYER_MASK_1`)

References

1 "Command/Response Unit - Programming ICD specification," Michael Huffer, LAT-TD-01547.

2 "LAT Communications Board - Programming ICD specification," Michael Huffer, LAT-TD-00860.

3 "GASU Based Teststands - A hardware and software primer," Michael Huffer, LAT-TD-03664.

4 "Event Builder Module - Programming ICD specification," Michael Huffer, LAT-TD-01546.

5 "GLT Electronics Module - Programming ICD specification," Michael Huffer, LAT-TD-01545.

Note: For additional resources, refer to the LAT Electronics, DAQ Critical Design Requirements List. On the LAT Electronics, Data Acquisition & Instrument Flight Software page (http://www-glast.slac.stanford.edu/Elec_DAQ/Elec_DAQ_home.htm), click Hardware and then click List of all documents.

Document Control Sheet

Table 1 Document Control Sheet

Document	**Title:**	LAT Inter-module Communications A reference manual		
	Version:	2.2		
	Issue:	2		
	Edition:	English		
	ID:	LAT-TD-00606		
	Status:	Draft - for internal distribution only		
	Created:	February 9, 2002		
	Date:	July 14, 2004		
	Access:	V:\GLAST\Electronics\Design Documents\LATcomm\2.2\frontmatter.fm		
	Keywords:	LAT Inter-module Communications, LAT comm		
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		

Table 2 Approval sheet

Name	Title	Signature	Date
Gunther Haller	LAT Chief Electronics Engineer		
JJ Russell	Flight Software Lead		

Document Status Sheet

Table 3 Document Status Sheet

Title:	LAT Inter-module Communications A reference manual		
ID:	LAT-TD-00606		
Version	**Issue**	**Date**	**Reason for change**
0.0	1	2/09/2002	Initial draft
1.0	1	2/28/2002	Initial internal draft to support release of “TEM - A Primer”
1.1	1	3/13/2002	Response from GXH. Initial draft sent to subsystems to support release of “TEM - A Primer”
1.2	1	4/02/2002	Added event shape picture for Dave L.
1.3	1	4/03/2002	Mirror actual changes in hardware as TEM is built: 1. Start bit from trigger message now included in event summary 2. Sense of zero suppression bit in trigger message changed (also reflected in event summary)
1.4	1	4/03/2002	Moved TACK field in trigger message to match hardware.
1.5	1	4/12/2002	Corrected inadequate english and typos as mentioned in C. Houchen’s (GSFC) e-mail of April 5.
2.0	1	11/06/2002	Changed overall event format
2.1	1	05/07/2003	Much needed revision of document.
2.2	1	03/12/2004	Updated fonts.
2.2	2	06/04/2004	Corrected some typos. Updated References.

Table of Contents

Abstract 3

Hardware compatibility 3

Intended audience 3

Conventions used in this document 3

References 4

Document Control Sheet 5

Document Status Sheet 6

List of Figures 9

List of Tables 11

Chapter 1 **LAT Protocol** 13
- 1.1 Introduction 13
- 1.2 Physical signalling 14
- 1.3 Cells and framing 15
- 1.4 Addressing 18
- 1.5 The header 19
  - 1.5.1 Usage of the protocol fields 20
- 1.6 Receiving a packet 21
  - 1.6.1 Calculating header parity 21
  - 1.6.2 Calculating data parity 22
- 1.7 Flow control 22
  - 1.7.1 The pause signal and truncate field 23

1.7.1.1 Discard . . . 24
1.7.1.2 Resend . . . 24
1.7.2 Flow control and broadcasting . . . 24
1.8 Behaviour on reset . . . 25
1.9 Statistics . . . 26
1.9.1 Transmitter statistics . . . 26
1.9.2 Receiver statistics . . . 26

Chapter 2 **Commanding** . . . 29
2.1 Introduction . . . 29
2.2 Abstract communication model . . . 30
2.3 The Command/Response fabric . . . 30
2.3.1 Commanding and Mastership . . . 30
2.3.2 Responders . . . 31
2.3.3 The CRU . . . 31
2.4 Command/Response Protocol . . . 32
2.5 Path redundancy and the Look-At-Me command . . . 33
2.6 Setting up the fabric . . . 35
2.6.1 Designating a commander . . . 37
2.6.2 Designating a commander . . . 38
2.6.3 Selecting the internal nodes . . . 39
2.6.4 Selecting the external nodes . . . 41
2.6.5 Node Assignments . . . 42
2.7 Register model . . . 44
2.8 Command string . . . 45
2.8.1 The Function field of the access descriptor . . . 46

Chapter 3 **Events** . . . 47
3.1 Introduction . . . 47
3.1.1 Generic field definitions . . . 47
3.2 Event Transport . . . 47
3.2.1 Protocol basis . . . 48
3.3 Event shape . . . 48
3.4 Contribution shape . . . 49
3.5 The Event Summary . . . 52
3.6 Error processing . . . 53

Chapter 4 **The Trigger** . . . 55

List of Figures

Figure 1 p. 13 The LATp Building block
Figure 2 p. 15 A cell
Figure 3 p. 16 A cell, constructed with either byte or bit-wide LATp
Figure 4 p. 16 A delineator
Figure 5 p. 17 An isolated, single-cell packet “on the wire”
Figure 6 p. 18 Two back-to-back single cells packets “on the wire”
Figure 7 p. 18 A multi-cell packet with one control cell and one data cell “on the wire”
Figure 8 p. 19 Structure of a LATp node address
Figure 9 p. 20 Structure of the header for a control cell
Figure 10 p. 21 Abstract receiving engine
Figure 11 p. 25 The pause signal and broadcasting
Figure 12 p. 26 Transmitter statistics register
Figure 13 p. 26 Receiver statistics register
Figure 14 p. 29 Command/Response fabric abstraction
Figure 15 p. 30 Interface Model for Command/Response protocol
Figure 16 p. 32 The Command/Response fabric
Figure 17 p. 34 Abstract model of path selection
Figure 18 p. 35 Encoding of the Look-At-Me (LAM) command
Figure 19 p. 36 Redundant nodes of the Command/Response fabric (internal to GASU)
Figure 20 p. 37 Redundant paths of the Command/Response fabric (external to GASU)
Figure 21 p. 38 Redundant paths of the Command/Response fabric (external to GASU)
Figure 22 p. 39 Redundant paths of the Command/Response fabric (external to GASU)
Figure 23 p. 40 Command/Response fabric after selection of CRU

Figure 24 p. 41 Command/Response fabric after selection of internal responders
Figure 25 p. 42 Redundant paths of the Command/Response fabric (external to GASU)
Figure 26 p. 45 Register Model
Figure 27 p. 45 Command *string*
Figure 28 p. 48 Event Fabric topology
Figure 29 p. 50 Natural contribution
Figure 30 p. 50 Diagnostic contribution
Figure 31 p. 51 Natural contribution in error
Figure 32 p. 51 Diagnostic contribution in error
Figure 33 p. 52 Structure of the Event summary

List of Tables

Table 1 p. 5 Document Control Sheet
Table 2 p. 5 Approval sheet
Table 3 p. 6 Document Status Sheet
Table 4 p. 17 Delineator values
Table 5 p. 20 Usage of the protocol field for the Command/Response fabric
Table 6 p. 42 Assignment of master node address on the Command/Response fabric
Table 7 p. 43 Assignment of slave node address on the Command/Response fabric
Table 8 p. 46 Enumeration of function field

Chapter 1 LAT Protocol

1.1 Introduction

The protocol model assumes as a basic building block, transmitter and receiver pairs. Each pair is connected through *three* abstract signals:

data: The `data` signals are used to transmit information *from* transmitter to receiver. Information is encoded with the protocol described below. The data signals may be either bussed or carried point-to-point. The *width* of this signal can vary depending on implementation. At this time, there are two options:

— *Bit*-wide. Information is carried on *one* data signal.

— *Byte*-wide. Information is carried on *eight* data signals.

pause: The `pause` signal is used to facilitate flow control. The receiver *asserts* this signal and the transmitter *monitors* this signal. The `pause` signal *must* be carried point-to-point. The issue of flow control and its implementation are discussed in Section 1.7. Depending on the flow-control model used, this signal is *optional*. For example, it is not used by the CRP protocol. (See 2.4.)

sysclk: The protocol assumes that both transmitter and receiver receive a common system clock (`sysclk`). This clock runs at a nominal frequency of 20 MHZ with a balanced, 50% duty-cycle. As LATp is an asynchronous protocol, its operation is insensitive to the actual `sysclk` rate.

In addition, it is assumed both the transmitter and receiver must be able to respond to an external ***reset*** signal. This signal may be driven in common to the two blocks, or the signal to each block may be driven individually. The building-block model is illustrated in Figure 1:

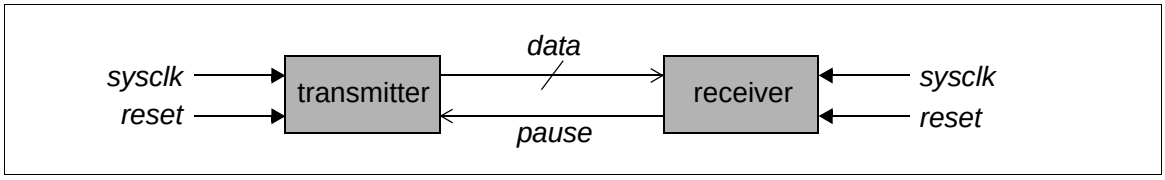

Figure 1 The LATp Building block

Any protocol participant is called a ***node***. A set of nodes and their respective interconnections is called a ***fabric***. The LAT contains more then one fabric, each fabric tailored to meet a different set of communications requirements, with varying number of nodes, flow-control models and connectivity. A module may be a participant on more then one fabric. For example, the TEM participates on both the CRP and event fabrics.

A node constitutes the addressable entity of a fabric. At a minimum, a node consists of either a transmitter or a receiver. Such a node is said to have a *half-duplex* connection to its fabric. A half-duplex node which consists of only a receiver is called a *slave*. A half-duplex node which consists of only a transmitter is called a ***master***. Alternately, a node may consist of both a receiver *and* transmitter. Such a node is said to have a *full-duplex* connection to its fabric.

The protocol provides the appearance of transporting information to and from its nodes in units of ***packets***. Packets are composed of one or more irreducible units of information called *cells*. The first cell of any packet is called a *control* cell. The first 16 clocks of a control cell contain a structure called the cell's *header*. The header's principal function is to provide packet routing; for example, the header contains a packets's destination address. The header's structure and purpose is discussed in detail in Section 1.5. Cells which follow a control cell, but without a header, are called *data* cells.

In short, a packet is composed of at least one control cell and has zero, one, or more data cells. Packets are sent to and received by one of the addressable entities of a fabric. An addressable entity on the fabric is called a node.

1.2 Physical signalling

Although not strictly necessary from the viewpoint of LATp, in practice, both `data` and `pause` signals are carried "on the wire" differentially, using LVDS (Low Voltage Differential Signalling). This implies that the `pause` signal requires *two* wires and the `data` signal requires 2 *x n* wires, where *n* is equal to the implementation's data width.

The commercial parts used within the LAT guarantee a stable known state of `HIGH`, for floating, terminated, or shorted receiver inputs. Consequently, in order to have a failed component appear quiescent on the wire, negative logic is used, where depending on usage:

HIGH: Implies a *false*, *clear*, or *zero* value.

LOW: Implies a *true*, *set*, or *one* value.

Typically, both clock and reset signals also have their origin in LVDS; however, it is likely these signals are shared with other elements of a design, and therefore, their physical signalling mechanism is implementation dependent.

1.3 Cells and framing

Information is clocked on the `data` lines at the system clock rate (`sysclk`). Any information requiring decoding by the protocol is called *control* information. Any information which does *not* require decoding is called *data* information. Both the amount of information transmitted per clock and its representation depend on the implementation *width*. The protocol supports two widths:

Bit-Wide LATp: The data signal is one bit wide; consequently, information is transmitted *one* bit per clock. The representation of control and data information is identical.

Byte-wide LATp: The `data` signal is one *byte* wide; consequently, information is transmitted *eight* bits per clock. The eight `data` signals are numbered *zero* through *seven*. Data information uses all eight of the `data` signals. Control information uses only the low order (*zero*) data signal. The remaining seven data signals must be *zero*.

In short, control information, independent of implementation width, has only *one* significant bit and when present in a byte-wide implementation is contained on the low-order `data` signal. Data information, on the other hand, always occupies *all* `data` signals.

An *idle* transmitter will clock *zeros* on all `data` signals. Information is both transmitted and received Most Significant Bit (MSB) *first*. Information is transmitted in units of *cells*. The cell is the irreducible unit of communication on a fabric. A cell is 130 clocks long and is composed of 128 clocks of data, followed by 2 clocks of control, as illustrated in Figure 2:

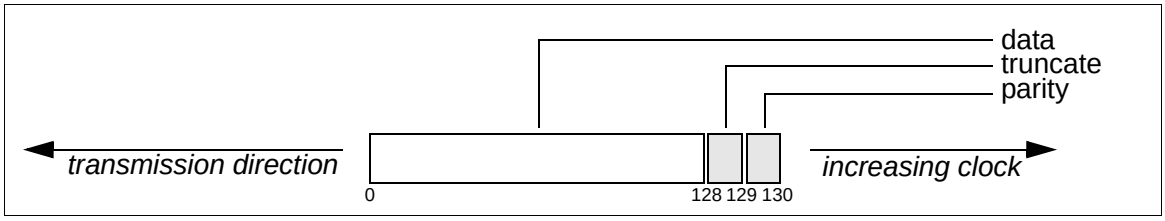

Figure 2 A cell

As data size varies with width, a cell constructed using *bit*-wide LATp has a *data* size of 128 *bits*, while a cell constructed using *byte*-wide LATp has a size of 128 *bytes*. However, as control size does not vary with width, all cells have an identical control size, which is 2 *bits*. To illustrate, Figure 3 shows the same cell constructed with either *byte*-wide LATp or *bit*-wide LATp.

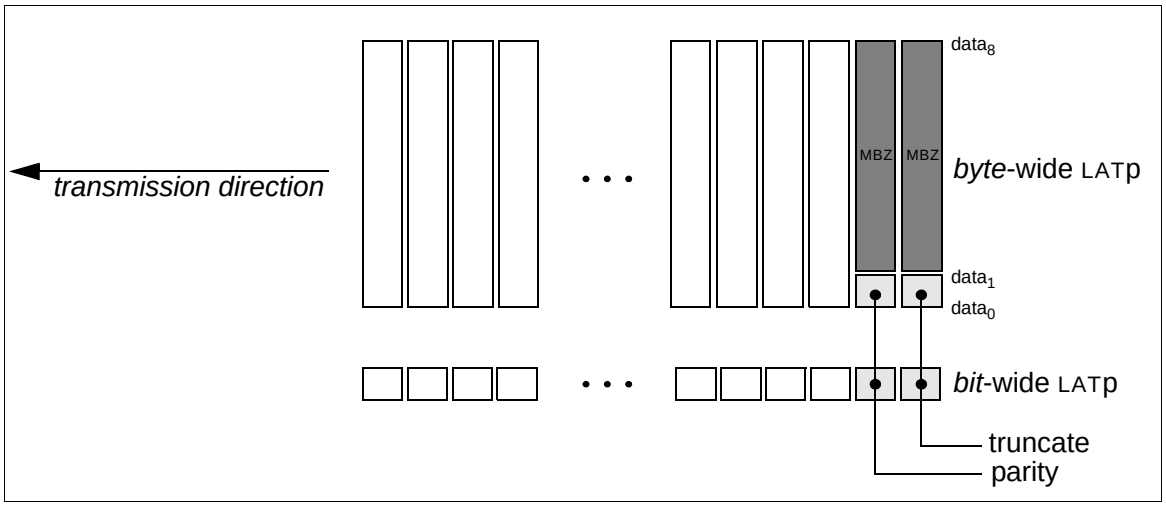

Figure 3 A cell, constructed with either *byte* or *bit*-wide LATp

truncate: The truncate control field, which occurs on the 128th clock, reflects the state of the `pause` signal which is discussed in more detail in Section 1.7.

parity: The parity control field, which occurs on the 129th clock, always contains the *odd* parity of the preceding *129* clocks worth of data. The use of the parity field is described in Section 1.6.2.

Cells are differentiated by whether the protocol decodes their sixteen most significant bits. Such cells are called ***control*** cells, and the decoded word is called the cell's *header*. Cells which only have their ***parity*** field decoded are called *data* cells.

The presence of a cell on the wire is always announced by a 2-clock control field immediately preceding the cell. This field is called a *delineator*. The value of the bit corresponding to the first clock of the delineator announces the start of a cell. If this bit is *set*, a cell is assumed to immediately follow the delineator. The value of the bit corresponding to the second clock of the delineator specifies whether the following cell is either a control or data cell. If the bit is *set*, the cell is a control cell. If the field is *clear*, the cell is a data cell. The encoding of a delineator is illustrated in 4:

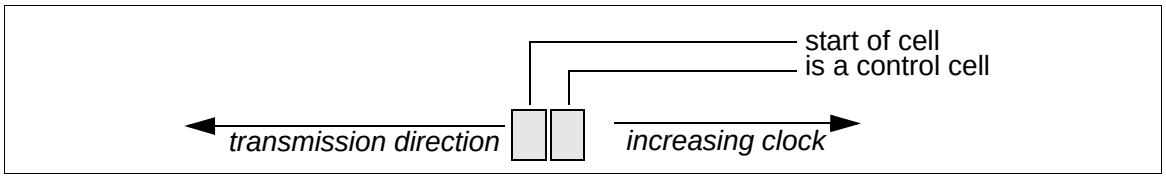

Figure 4 A delineator

Alternately, a delineator may also be considered as representing one of four values as enumerated in Table 4.:

Table 4 Delineator values

Value		Description
binary	**decimal**	
11	3	Start of packet
10	2	Start of payload
01	1	Reserved (not used)
00	0	End of packet

Control cells may be sent in isolation; however, all data cells must be preceded by a control cell. If cells are announced by a delineator, then by analogy, data cells may be thought of as *announced* by a control cell. A *packet* is defined as set of related, contiguous cells. A packet has one control cell, followed by zero, one, or more data cells. For example, a one cell packet encoded "on the wire," would consist of a single control cell as illustrated in Figure 5. This cell is prefixed with a delineator of value *three*, announcing the presence of a control cell, and it is terminated with a delineator of value *zero*. The data length of the packet would be 14 bytes if sent bit-wide and 126 bytes if sent byte-wide.[1]

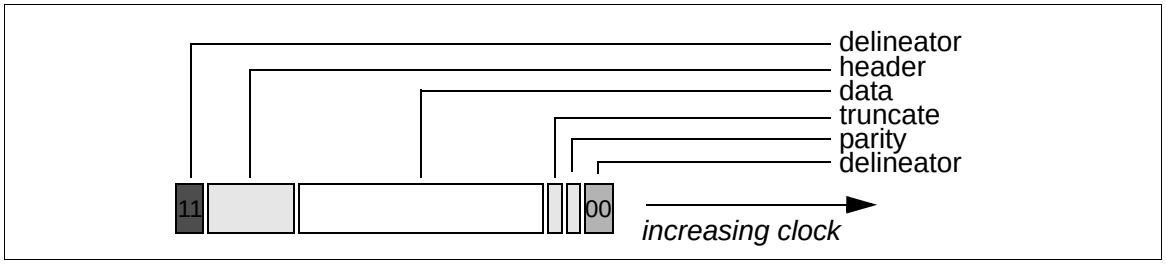

Figure 5 An isolated, single-cell packet "on the wire"

If two *different* fixed-length packets were sent "back-to-back," they would have an "on wire" structure as illustrated in Figure 6:

1. Less two bytes, to account for the header of the control cell.

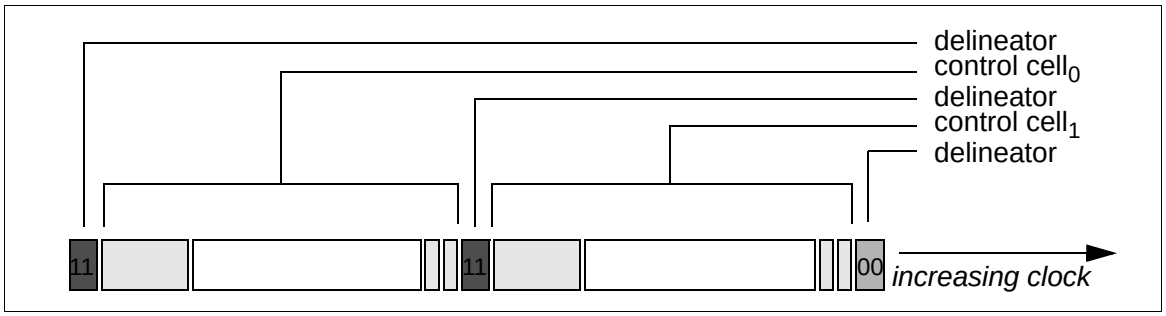

Figure 6 Two back-to-back single cells packets “on the wire”

Sending a variable length packet requires at least two cells, one and only one control cell, and one or more data cells. For example, transmitting a two-cell packet would be accomplished with the combination of one control cell and one data cell. This packet requires three delineators: the first (with a value of *three*) to announce the packet, the second (with a value of *two*), to differentiate the packet’s two cells, and the last (with a value of *zero*) to terminate the packet. The “on wire” structure for this packet is illustrated by Figure 7:

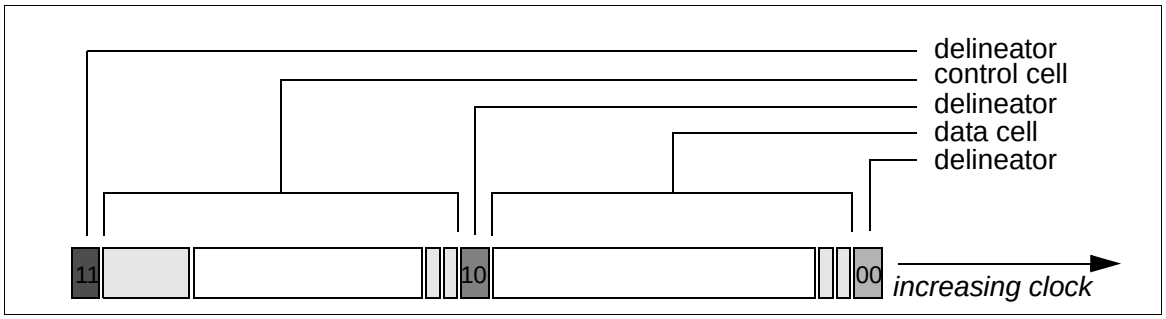

Figure 7 A multi-cell packet with one control cell and one data cell “on the wire”

1.4 Addressing

LATp specifies a six-bit address space, theoretically allowing for up to sixty-four nodes on any given fabric. However, in actual fact, this number is subject to some constraints as:

- The address space is (lightly) encoded.
- One address is reserved for broadcast operations.

The structure of a node address is illustrated in Figure 8:

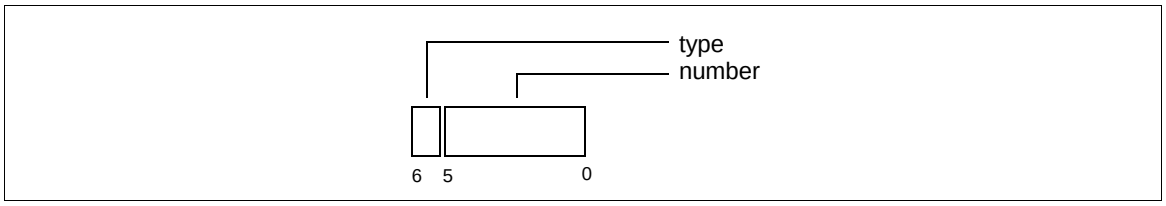

Figure 8 Structure of a LATp node address

type: A one-bit field which allows a node address to be broken up into two arbitrary spaces. This field is used for broadcast operations as described in Section 1.6.

number: A value from 0 to 1E (hexadecimal) which is used to help differentiate nodes on the fabric. The value 1F (hexadecimal) is reserved and called the *broadcast* number.

In order to participate on a fabric, a node is assigned *a priori* both a type and unique five-bit number. This combination of type and number is called a node's *own* address. The combination of a node's type and the broadcast number is called a node's *own* broadcast address. To summarize: from the protocol's perspective the only constraints on address assignment are:

a. Each node on a fabric must have a unique node *address*.

b. Node *number* 1F (hexadecimal) is reserved for broadcast operations.

It is at the implementation's discretion whether or not to respond to its own *broadcast* address.[1] The broadcast address is intended to allow collective addressing of all nodes of the same *type*; consequently, there is no operation defined to allow collective addressing of *all* nodes on a fabric, independent of type.

1.5 The header

The header is considered control information and is 16 bits in length. Thus, independent of implementation width, 16 clocks are required to either clock in or clock out its information. The header's principal function is to provide routing information for its packet. It also serves as a bridge to the data information of its corresponding packet, allowing the client to describe abstract attributes to packet data, independent of the data itself. The sixteen bits of the header are encoded as illustrated in Figure 9:

1. For example, on the CRP fabric, the AEM will *not* respond to its own broadcast address. (See 2.3.)

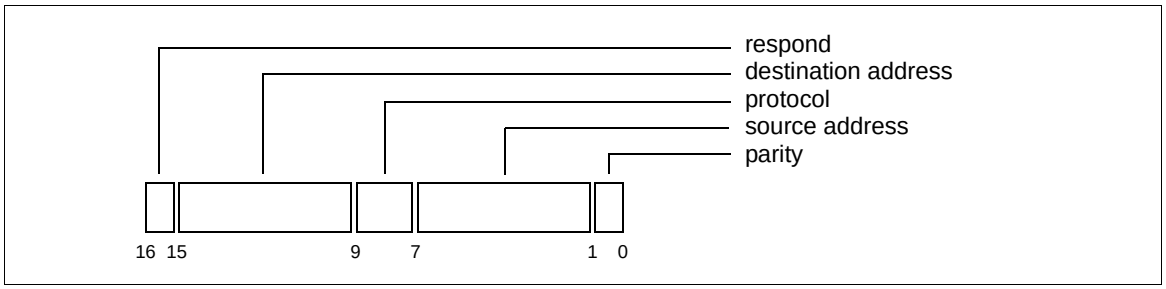

Figure 9 Structure of the header for a control cell

respond: This field determines whether the packet's sender expects a packet to be sent in response to receiving this packet. If the bit is *set*, a response is expected. If the field is *clear*, no response is expected. If a response is expected, the destination address of the response is contained in the ***source address*** field (described below).

destination address: This field contains the address of the node to receive the packet. Legitimate values for this field are discussed in Section 1.4.

protocol: This field is used to enumerate how packet data is encoded. Its values are *not* set or used by *this* protocol. Instead, they are meant to be used and interpreted by *clients* of this protocol. See Section 1.5.1, for an example on how the values of this field are assigned.

source address: This field contains the address of the node which originated the packet. Legitimate value for this field are discussed in Section 1.4.

parity: The *odd* parity over the entire 15-bit length of the header. See the following section for a description on how this field is used.

1.5.1 Usage of the protocol fields

Table 5 Usage of the protocol field for the Command/Response fabric

Protocol value[1]	Duplex	Description
00		Reserved
01		Reserved
10		Reserved

1. Binary

1.6 Receiving a packet

In its most abstract form, as illustrated in Figure 10, the receiver is in one of two states: *idle* or *receiving*:

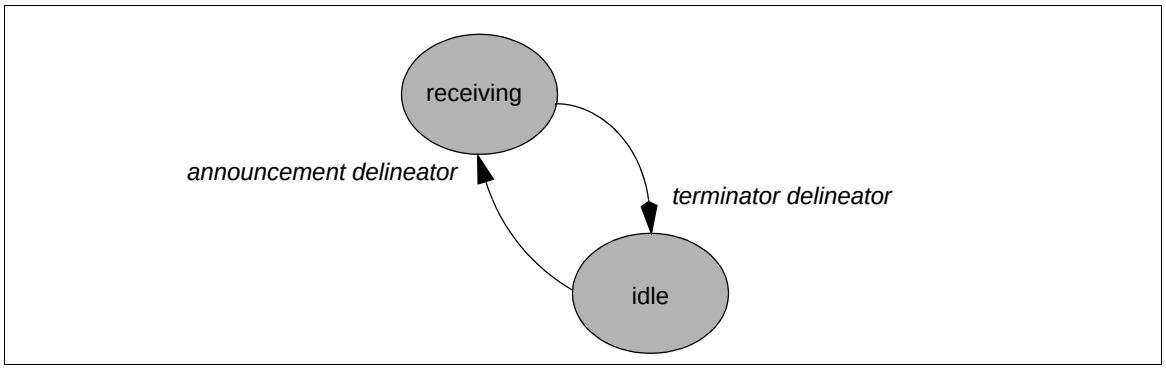

Figure 10 Abstract receiving engine

While idling, the receiver is waiting on the arrival of an announcement delineator. Once detected, the receiver is committed to clocking cells of the corresponding packet off the wire until reaching a termination delineator. While clocking cells of the wire, the receiver is said to be *receiving*. The receiver's primary responsibility while in this state is determining whether or not it is the recipient of the transmitted packet. If not, the cells off the packet are simply discarded. The receiver determines whether it is the target of the packet by decoding the destination address contained in the header of the packet's control cell. (See Section 1.5.) The receiver matches this address against either the receiver's own address or against its own broadcast address. If the match is successful, the packet will be delivered to the receiver's client. However, before delivering the packet the receiver must:

- Update the receiver's ***packets received*** counter. (See Section 1.9.2.)
- Check data integrity on both header and all data cells. (See Section 1.6.1.)

Once these tasks are accomplished, the packet is passed to its client along with the results of the data integrity check.

1.6.1 Calculating header parity

The 16^{th} clock of the header of any control cell always contains the *odd* parity of the preceding 15 bits. This value is called the header's *parity*. (See Section 1.5.) As the receiver consumes a control cell it also calculates header parity. The receiver compares its calculation against the header's parity. If this comparison fails, the receiver has the following responsibilities:

- *Set* the ***header parity error*** field of its receiver statistics register. (See Section 1.9.2).
- *Discard* the cells following the control cell. The *data* parity (see Section 1.6.2) of the control cell is *not* checked.

- *Signal* the client that a header parity error has occurred.

If the comparison succeeds, the data parity of the control cell and any subsequent data cells must be validated as described in the following section.

1.6.2 Calculating data parity

The 129th clock of the header of any cell (including a control cell) always contains the *odd* parity of the preceding 129 clocks. This value is called the cell's *parity*. (See Section 1.3.) As the receiver consumes a cell it also calculates the cell's parity. The receiver compares its calculation against the cell's parity. Note that this calculation is width dependent. If this comparison fails, the receiver has the following responsibilities before passing the packet on to its client:

- *Set* the ***data parity error*** field of its receiver statistics register. (See Section 1.9.2.)
- *Discard* the cells following the erring cell.
- *Signal* the client that a cell parity error has occurred.

1.7 Flow control

LATp makes no assumptions on whether a receiver can *sink* cells at the same rate its transmitter can *source* cells. In other words, the protocol considers a transmitter which over-runs its receiver an acceptable (albeit, an infrequent and transitory) condition. Consequently, the protocol must provide a ***flow-control*** mechanism to deal with condition, when and if it appears. In fact, in order to encompass the different requirements imposed by the various places within the LAT where the protocol is employed, LATp specifies *three* different flow-control models. While different, each of the models are based on four common premises:

a. While a receiver is *busy*, its corresponding transmitter cannot send. In such a condition, the transmitter is said to be *paused*.

b. If a receiver becomes busy while its transmitter is sending a packet, the transmitter is obligated to stop. This is referred to as *truncating* the packet currently being sent.

c. If packets are truncated, they will always be truncated on cell boundaries.

d. While busy, a receiver must retain enough capacity to absorb *one* additional cell.

These models are:

None: This is the degenerate case of the four premises enumerated above. In this model, a receiver is by definition *never* busy, as a transmitter only sends and the receiver only accepts *one*-cell packets. In such a case, the `pause` signal is not necessary and may be omitted. The CRP fabric (see 2.3) is an example of this flow-control model.

Discard: When a receiver is busy, it asks its corresponding transmitter (through the PAUSE signal) to stop sending. The transmitter is not permitted to send while the receiver is busy. If the receiver asks the transmitter to stop *while* sending a packet, the packet being sent will be truncated. The remainder of the packet will be *discarded*. The movement of event contributions from electronics modules (for example, the TEM) to Event Builder (see [1]) uses this flow-control model.

Resend: When a receiver is busy, it asks its corresponding transmitter (through the PAUSE signal) to stop sending. The transmitter is not permitted to send while the receiver is busy. If the receiver asks the transmitter to stop *while* sending a packet, the packet being sent will be truncated. The remainder of the packet will be *resent*. However, the remainder will be sent as a *new* packet. Event movement from Event Builder to LCBs (see [1]) uses this flow-control model.

As is apparent, the latter two models share many common features and only differ in what they do with the remainder of a discarded packet. These two models are discussed in more detail in the following section.

1.7.1 The pause signal and truncate field

Comment: GXH points out that this mechanism assumes the transmitter can respond to changes in the `pause` *line within one clock cycle and if not, may be a hole when sending last bit of a cell. Should be thought about (later)...*

Flow control for both Discard and Resend models are based on the usage of the `pause` signal and a cell's `truncate` field. (See Section 1.3.) Whenever a receiver decides it is in danger of being over-run, it *asserts* the `pause` signal. As long as the `pause` signal is asserted, the receiver is said to be *busy*. Any *positive* transition of the `pause` signal specifies to the outside world that the receiver has gone to the busy state. While it is at the implementor's discretion to determine what constitutes busy, any implementation must guarantee, at a minimum, enough receive capacity after asserting *pause* to absorb *one* additional cell. Consequently, all implementations must go busy when they cannot successfully absorb at least one cell. For example, a typical receiver implementation may buffer incoming packets in a FIFO. A very reasonable derivation of the `pause` signal would use the "almost-full" flag of this FIFO. The FIFO would assert this flag whenever it does not have enough buffering for at least one cell.

From the perspective of the transmitter a receiver enters the busy state spontaneously, that is, either *during* or *outside* packet transmission. The transmitter is continually monitoring the state of the receiver using the *pause* signal. While the receiver is *not* busy, the value of the ***truncate*** field of any cells transmitted must be *clear*.

When the transmitter detects that its receiver has gone busy, the transmitter goes to a *paused* state. While in a paused state, a transmitter's responsibilities depend on whether or not it is currently sending a packet. If *not* currently sending packets, the transmitter simply defers transmission until the *pause* signal is de-asserted. Once the receiver announces it is no longer

busy (by de-asserting the `pause` signal), the transmitter is free to recommence sending packets.

If the transmitter goes to the paused state while transmitting, the situation is somewhat more complex. First, the cell currently being transmitted is allowed to complete. The transmitter must then stop transmission until the `pause` signal is de-asserted. The receiver will successfully consume this cell, because (as mentioned previously) the receiver has guaranteed enough capacity to absorb one more cell after it asserts pause. This cell is called the *truncating* cell. The transmitter establishes the value of the ***truncate*** field of the truncating cell as follows: If the cell is *not* the last cell of the packet, its ***truncate*** field is *set*. If the cell *is* the last cell of the packet, its ***truncate*** field is *cleared*.

Once the truncating cell is sent and while the receiver is busy, the receiver is assured the transmitter will not send packets. At this point, both flow-control models assure:

- The receiver can, on demand, halt transmission of packets.
- Both transmitter and receiver can gracefully pause and resume packet transmission independent of each other's state.
- The receiver's *client* can determine, using just the packet, whether or not the receiver went busy while a packet was in transmission.

The question remains of how to deal with the remaining (or runt) cells of the truncated packet. It is on this issue that the two remaining flow-control models diverge.

1.7.1.1 Discard

This section needs work.

This solution depends on the following:

-

The simplest and most straight-forward way to deal the runt cells is to have them discarded by the transmitter. This is the solution used by the modules on the Event Fabric (see xxx).

1.7.1.2 Resend

To be written.

1.7.2 Flow control and broadcasting

Because the `pause` signal is carried point-to-point, the mechanics of flow-control when broadcasting differ little from their corresponding singlecast mechanics. (See Section 1.7.1.) The fundamental difference is that a common pause signal for the transmitter is developed by `OR`ing each receiver's pause signal as illustrated in Figure 11. Once developed, a transmitter uses this signal exactly as described in Section 1.7.1. A receiver's (flow control) behaviour on receiving a broadcast packet, whether or not it is truncated, does not differ from its behaviour

when receiving a singlecast packet. There are, however, two issues, one transmitter and one receiver related, that the implementor should be aware of with broadcasting:

- Because the transmitter, on broadcast, requires *all* its receivers to be available, the probability to successfully transmit (broadcast) packets now depends on the throughput of *all* its receivers, not just the throughput of any one receiver. Overall throughput could be affected by the relative frequency of broadcast to singlecast transmission. In turn, this may have an impact on the buffering required of a receiver.
- A receiver may receive a truncated packet which was not initiated by the receiver going to a busy state. Instead, it was because *another* receiver went busy. A receiver implementation must be very careful to decouple its busy processing from its handling of a truncated packet. Handling of a truncated packet should only depend on the state of a packet's truncate *field*.

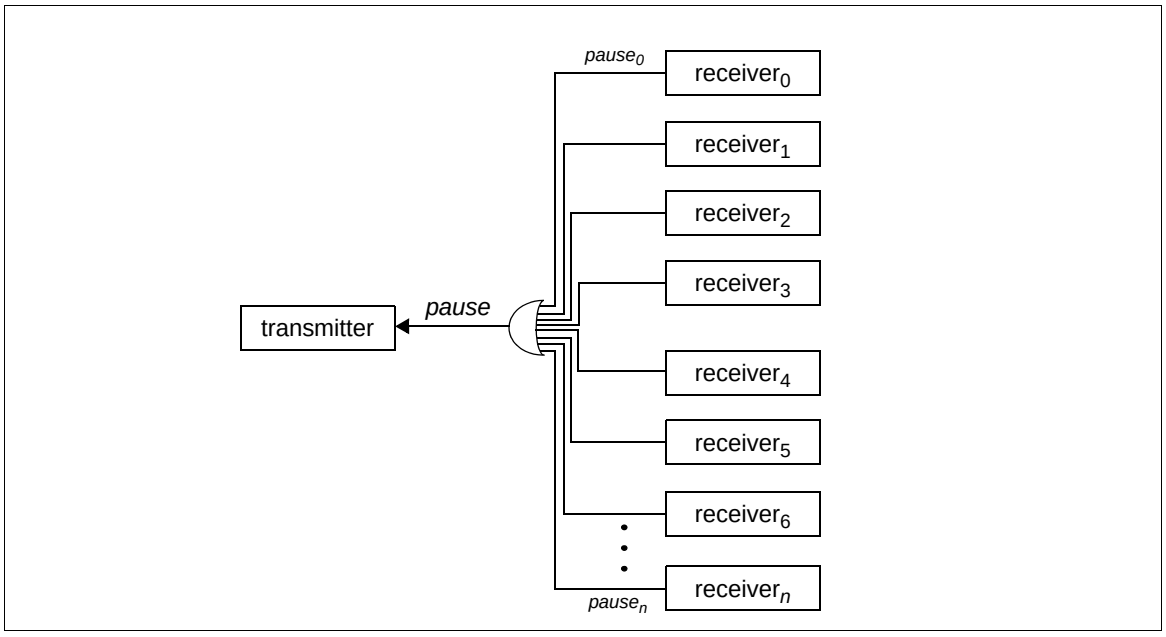

Figure 11 The pause signal and broadcasting

1.8 Behaviour on reset

Both transmitter and receiver must be sensitive to an external reset request.

On `reset`:

- The receiver will deassert the `pause` signal.
- The transmitter must terminate any transmissions.
- Any counters are set to zero.

1.9 Statistics

Each node monitors the health and state of communication between it and its fabric. The results of this monitoring are maintained in a predefined set of registers, one register for each receiver and one register for each transmitter. Access to these registers, independent of when their values change, must return a coherent and sensible value. There are two mechanisms to *zero* a counter:

a. Writing the register. The value to be written is ignored.

b. The receipt of an external `reset` signal.

The exact mechanism by which registers are accessed is outside the responsibility of the protocol and depends on implementation. For example, a typical implementation would provide access to these registers through the CRP protocol. (See 2.4.)

1.9.1 Transmitter statistics

Each transmitter maintains the sixteen-bit data structure described in Figure 12:

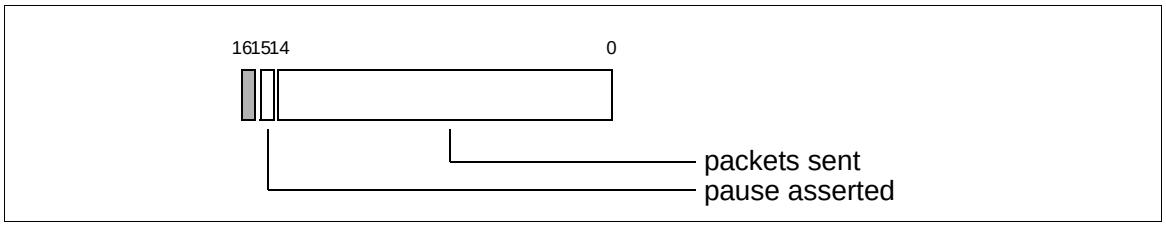

Figure 12 Transmitter statistics register

packets sent: The number of control cells successfully transmitted. The counter has a 14-bit range. It is saturating, *i.e.*, when the counter overflows it no longer increments until re-zeroed.

pause asserted: If the `pause` signal was ever asserted, this field is *set*.

1.9.2 Receiver statistics

Each receiver maintains the sixteen-bit data structure described in Figure 13:

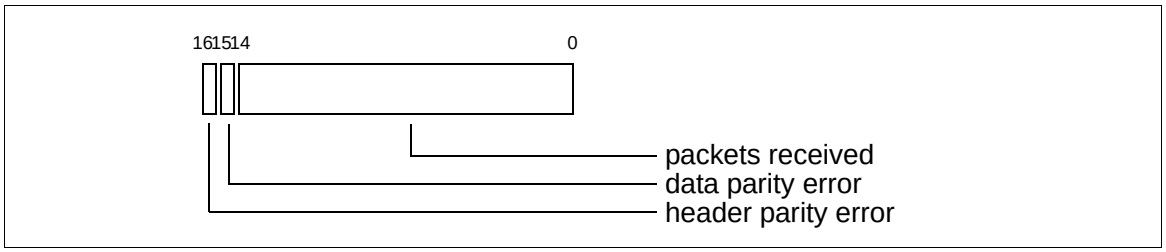

Figure 13 Receiver statistics register

packets received: The number of control cells whose destination address is successfully matched by the receiver. See Section 1.6 for a discussion on when this counter increments. The counter has a 14-bit range. It is saturating, *i.e.*, when the counter overflows it no longer increments until re-zeroed.

data parity error: This field summarizes any cell parity errors discovered by the receiver for any of the packets it has processed. If this field is *clear*, the receiver has not detected any cell parity errors. If *set*, the receiver has detected at least one cell parity error. See Section 1.6 for a discussion on the conditions under which the value of this field is changed.

header parity error: This field summarizes any header parity errors discovered by the receiver for any of the packets it has processed. If this field is *clear*, the receiver has not detected any header parity errors. If *set*, the receiver has detected at least one header parity error. See Section 1.6 for a discussion on the conditions under which the value of this field is changed.

Chapter 2 Commanding

2.1 Introduction

Each one of the various LAT electronics modules define and maintain their individual state through a set of *registers*. The principal function of the Command/Response Protocol (CRP) and its corresponding fabric is to provide centralized access to these registers for the purpose of configuration, monitoring, and control. The central module providing this supervision is called the fabric's *commander*. The modules containing the registers accessed by the commander are called the commander's *responders*. The commander and its set of responders constitute the *nodes* on the Command/Response fabric. This abstraction is illustrated in Figure 14:

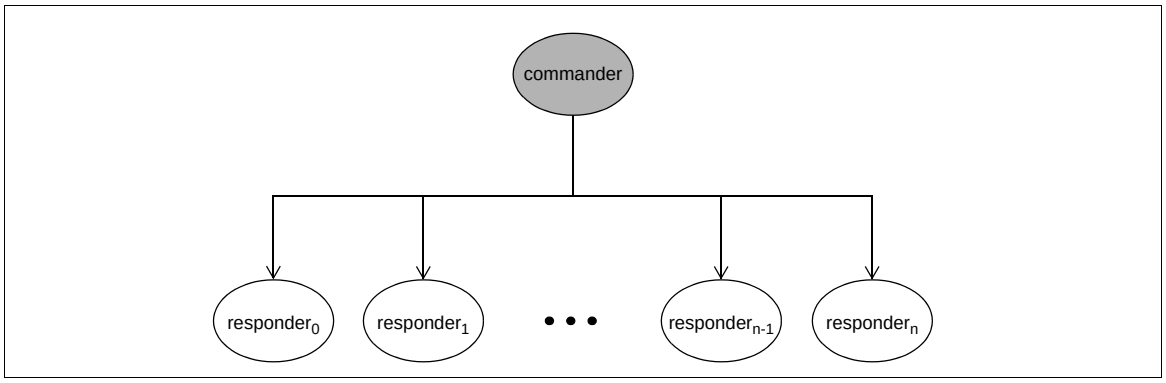

Figure 14 Command/Response fabric abstraction

The Command/Response fabric provides a mechanisms for a low-latency, deterministic exchange of information between command and responder, independent of other coincident LAT traffic such as event data and trigger communications. As this fabric is the principal means for communication within the LAT, it must also maintain high-availability. To this end, the fabric and its constituents have a high-degree of redundancy built-in, both in duplication of members and connectivity.

2.2 Abstract communication model

The model assumes that information is exchanged between commander and responder using four “wires.”

Physically signalled over each of these wires.

Each as illustrated in Figure 15:

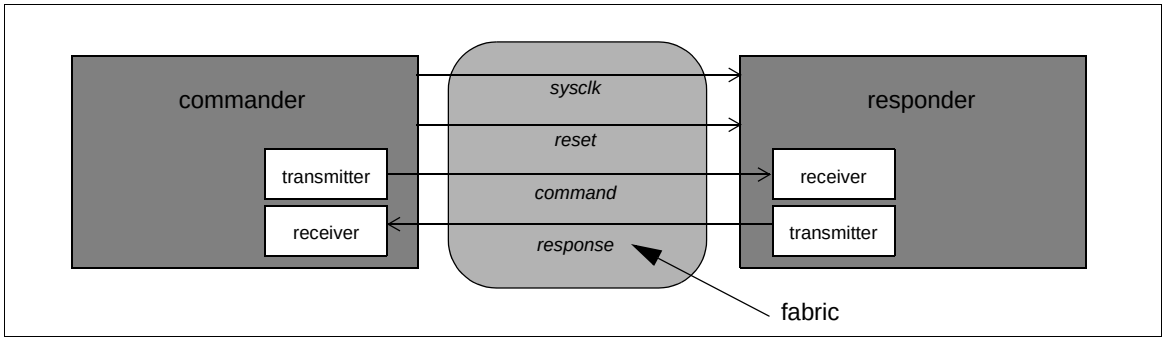

Figure 15 Interface Model for Command/Response protocol

sysclk:

reset:

command and response: Information on these wires is encoded using the LATp protocol described in Chapter 1. Note that depending on wire information is....

2.3 The Command/Response fabric

All nodes on the CRP fabric are divided arbitrary into two classes: those which are capable of *either* commanding or responding and those which may *only* respond. Nodes which may assume the role of either commander or responder are called *masters*. Those nodes which can only be responders are called *slaves*. This division is reflected in the value of the ***type*** field in a node’s own address. (See Section 1.4.) If the type field of a node’s own address is *set*, the node is a *master*. If the type field of a node’s own address is *clear*, the node is a *slave*. This implies that broadcast operations can be directed to either the set of all masters or the set of all slaves.

2.3.1 Commanding and Mastership

The LCB (LAT Communication Board) is designed to provide mastership capability on the fabric. (See [2].) Physically, the LCB is implemented as a 6U Compact PCI (cPCI) module. A node based on the LCB consists of a 4-slot cPCI crate, containing at a minimum[1] one SBC

1. Excluding, of course, the LCB.

(Single Board Computer). Software running in the SBC interacts with other nodes on the fabric through the LCB's PCI interface. A master node is commonly referred to as a "crate." Depending on its visibility outside the LAT, the fabric contains two types of crates:

EPUs: Event Processing Units. There are three EPUs: two are always active and one is envisioned as a "cold" spare. The principal function of an EPU is to process (or filter) events.

SIUs: Spacecraft Interface Units. Potentially, the LAT has up to three SIUs: two local and one (physically) external to the LAT. The external crate is used during LAT testing and commissioning to provide a means, independent of the spacecraft, to monitor and control the LAT during the commissioning phase. Typically, only one of the two local SIU's is envisioned as active, with the second SIU serving as a cold spare. The SIU, as its name implies, serves as an interface between LAT and spacecraft.

At any given time only *one* of these six crates may assume the role of commander. This crate is called the *designated* commander.

2.3.2 Responders

Those nodes which are slaves on the fabric include:

TEMs: The Tower Electronic Module. There are 16 TEMs, each servicing one tower.

GEM: The GLT Electronics Module. There are two: the *on*-board and *off*-board GEM; however, only one of the two is active at any one time.

AEM: The ACD Electronics module. There are two: the *on*-board and *off*-board AEM; however, only one of the two is active at any one time.

EBM: The Event Builder Module. There are two: the *on*-board and *off*-board EBM; however, only one of the two is active at any one time.

PDUs: Power Distribution Unit. There are *two*, and *both* may be active at any one time.

2.3.3 The CRU

The "glue" which binds together the nodes of the Command/Response fabric is the CRU (Command/Response Unit). Its essential function is to both "fan-in" and "fan-out" commands from the designated commander to its responders and to fan-in responses from responders back to the designated commander. The components of the CRU are organized along with the AEM, EBM, and GEM on a single Printed Circuit Board (PCB) called the DAQ board. The DAQ board resides in the GASU *box* as described in [3]. In actuality, in order to satisfy redundancy requirements, there are *two* DAQ boards, both identical and both residing in the same GASU box. One board is referred to as the *primary* DAQ board and the other as the *redundant* DAQ board. In typical operation only one of the two DAQ boards is powered. However, both boards *can* be powered simultaneously and different modules of the two boards can be "mixed and matched." Therefore, it is common practice to refer to the modules

sharing the same DAQ board as the CRU as its *on*-board modules and the modules residing on the alternate DAQ board as its *off*-board modules. The CRU on the primary DAQ board is called the *primary* CRU and the CRU on the redundant DAQ board, the *redundant* CRU. Only one of the two CRUs can operate at any one time; it can, however, be either the primary or redundant unit. A detailed description of the CRU is found in [1]. The fabric, its CRU and nodes are all illustrated in Figure 16:

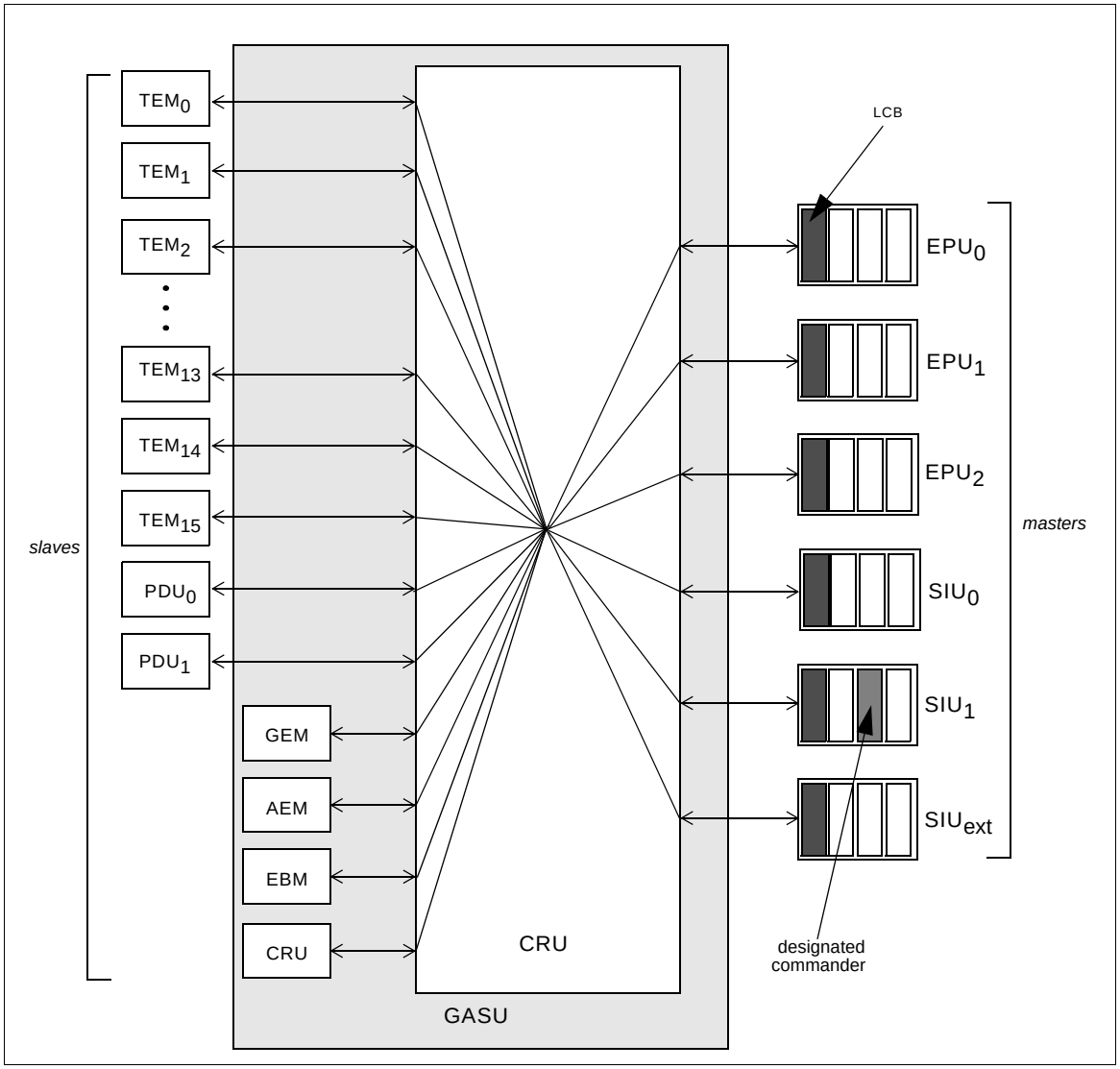

Figure 16 The Command/Response fabric

2.4 Command/Response Protocol

The protocol is a simple packet exchange between two entities using the ***command*** and ***response*** wires of the model described in Section 2.2. A commander directs a responder by

sending a packet on the command wire and a responder conditionally responds to the command by sending a packet on the response wire.

Both command and response packets are either sent or received using the *bit*-wide LATP protocol specified in Chapter 1. Both command and response packets are always just *one* cell long. As packet length is constrained to one cell, there is no need for flow control; therefore, the *pause* signal is neither necessary nor provided.

Only *command* packets are sent on the command path and only *response* packets are sent on the response path. Only the commander is capable of initiating packet communication. In short:

- A commander *transmits* on the command path and *receives* on the response path.
- A responder receives on the *command* path and transmits on the *response* path.
- The destination of a command may *only* be a responder. The destination of a response may *only* be a commander.
- A responder transmits *only* in response to a command. A commander transmits spontaneously.

In order to simplify the construction and operation of the fabric, only *one* of the six masters can command at any one time. This node is called the *designated* commander. It is the responsibility of software running on each of the masters to designate the particular node that will command. Commands requiring a response will be *timed-out* by the commander. The time-out value is determined by both the transmission time of a command to its destination and the expected delay by the responder in first issuing the response. Again, in order to simplify the fabric, only *one* transaction is allowed at a time between commander and responder. A transaction is defined as either the processing of one command, or if the command solicits a response, one command and its response.

The packet header for both commands and responses will have the protocol field set to *zero*.[1] Potential destination and source node addresses are enumerated in Tables 6 and 7. The MSB of an address will be *set*, if a node is a master. The MSB will be *clear*, if the node is a slave.

The encoding of the fourteen-byte data portion of the command or response cell is agreed on between commander and responder and varies by responder type. For example, the encoding necessary for a commander to access a TEM's registers is the same regardless of which TEM; however, a TEM's encoding differs from the encoding used to access the registers of a GEM. An example of such an encoding, in this case, for the CRU, can be found in [1].

2.5 Path redundancy and the Look-At-Me command

Each node on the Command/Response fabric maintains *two* sets of the four "wires" described in Section 2.2. One set is called the node's *primary* path and the other set, the node's *redundant*

1. There is one exception which is described in Section 2.5.

path. At any one time, the node uses only one of these two paths. This path is called the *selected* path. When the node is either powered on or reset (true?), the node, by default always selects the primary path. To toggle the currently selected path, a pre-defined CRP command is sent from commander to the node. This command is called the "Look-At-Me" (LAM). Commander and responder use this command as follows:

- The responder listens simultaneously on both primary and redundant paths for a LAM. The node determines that it's the target of a LAM in a fashion no different than it would in decoding any other CRP command. (See xxx and xxx.)
- The commander sends the LAM on *one* of the two paths. The path the commander chooses to send the LAM on (either primary or redundant) will become the selected path.
- The responder decodes the LAM and on the basis of which....

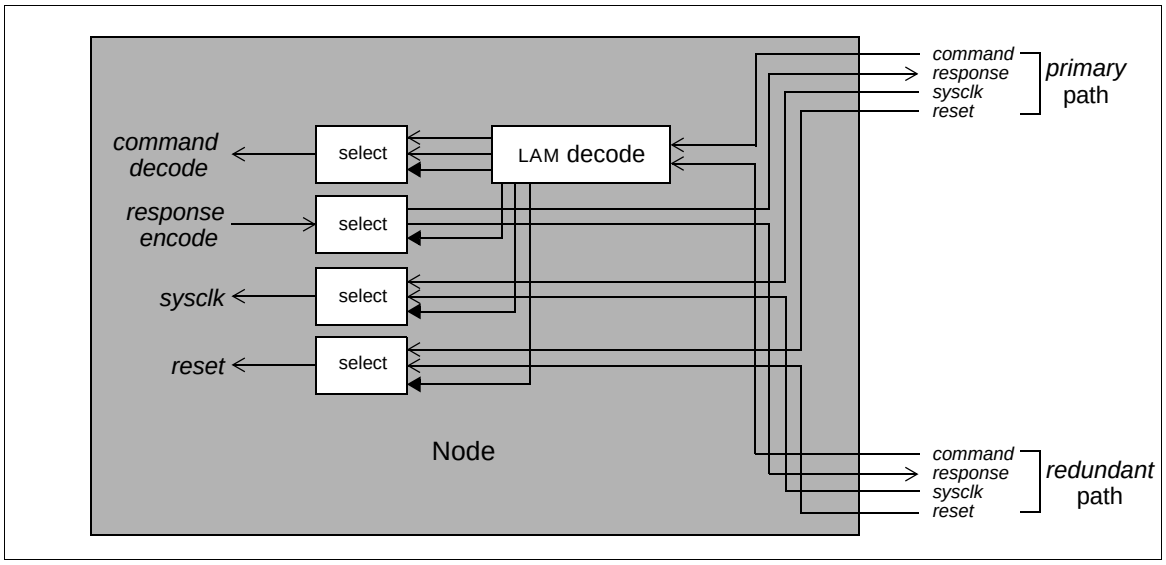

Figure 17 Abstract model of path selection

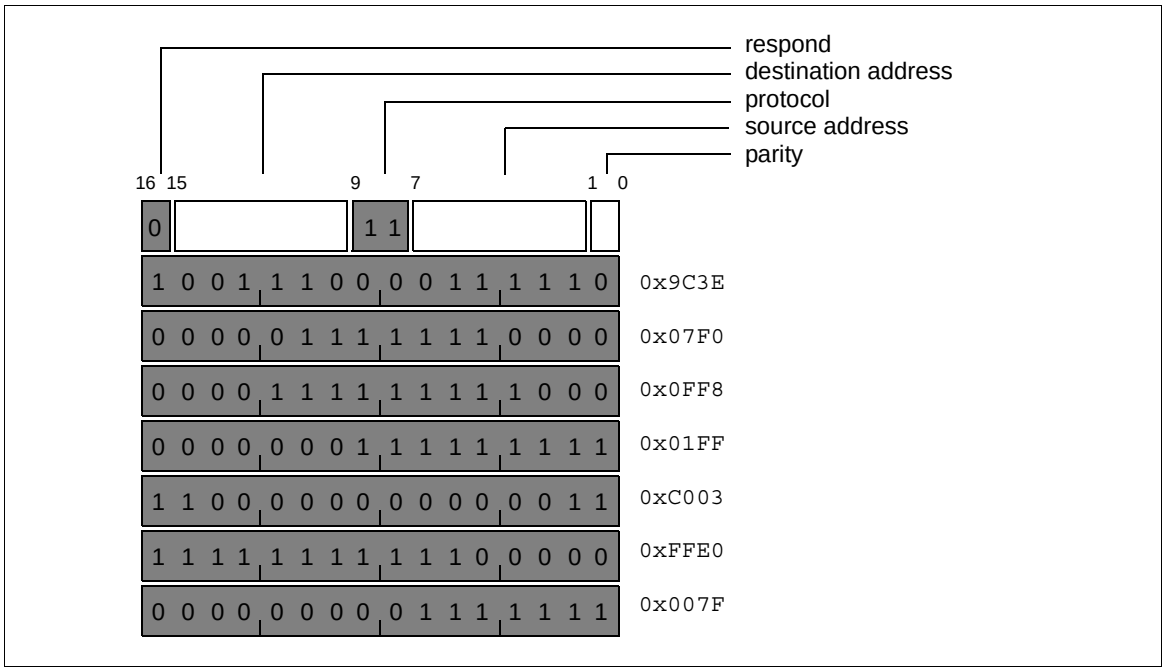

Figure 18 Encoding of the Look-At-Me (LAM) command

2.6 Setting up the fabric

In order to mitigate against single-point failure, a significant amount of redundancy is built into the fabric. All this redundancy is, unfortunately, configurable and one of the early and "one-time" responsibilities of the FSW system is to select and configure the minimum subset of components and paths necessary to use and operate the fabric. It is convenient, when discussing this setup strategy, to discuss redundancy selection for those nodes within the GASU (the *internal* nodes) and those nodes outside the GASU (the *external* nodes). In the former case, the nodes themselves are redundant; in the latter case, it is the *connections* to the nodes which are redundant.

Each GASU contains two DAQ boards. One board is called the *primary* board and the other the *redundant* board. Each DAQ board contains one CRU. The CRU on the primary board is called the primary CRU, and the CRU on the redundant board is called the redundant CRU. In addition to its CRU, each DAQ board contains three other modules: a GEM, AEM, and EBM. Each one of these modules has two connections,[1] one to its on-board CRU and the other to its off-board CRU. The desired goal is that by the end of the selection process only four out of the eight modules (one for each type) will participate on the fabric. The GASU provides enough

1. Where a connection is defined as described in Section 2.2 and includes the ***sysclk***, ***reset***, ***command*** and ***response*** wires.

connectivity between its two DAQ boards in order to mix and match modules from both boards as illustrated in Figure 19:

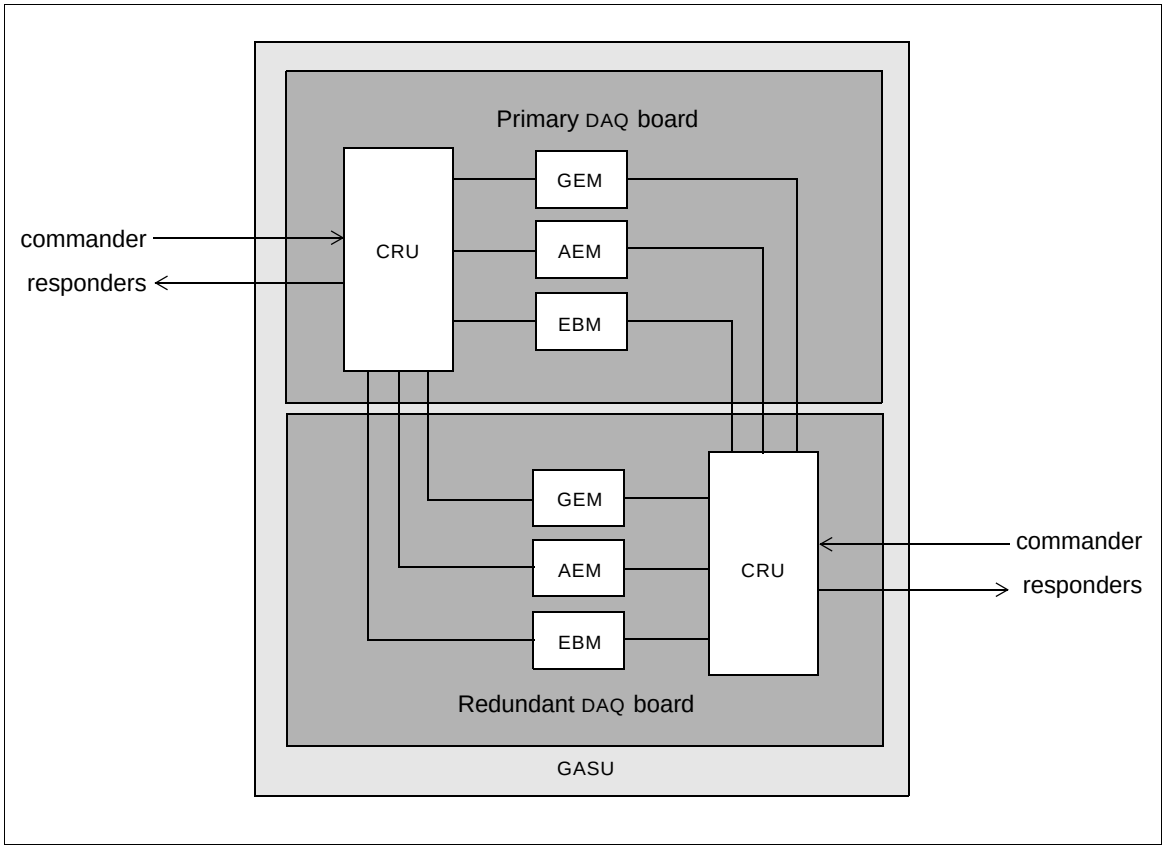

Figure 19 Redundant nodes of the Command/Response fabric (internal to GASU)

Each one of the nodes external to the CRU has two connections, one of which is redundant. One of the two connections is wired to the *Primary* CRU. This connection is called the node's primary *path*. The other connection is wired to the redundant CRU. This connection is called the node's redundant path. The desired goal is that by the end of the selection process only one of the two paths from each node will be connected to the fabric. The connectivity between external nodes and CRU is illustrated in Figure 20:

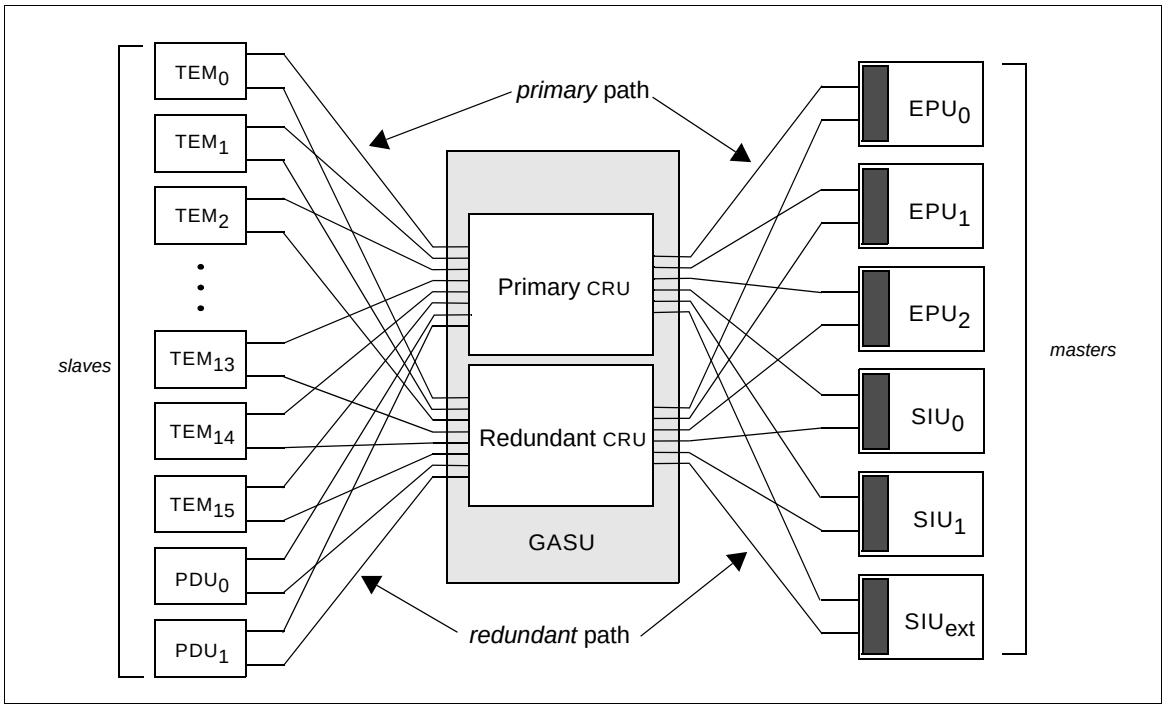

Figure 20 Redundant paths of the Command/Response fabric (external to GASU)

Given this background, there are five steps to configure the fabric:

a. Designate a commander.

b. Select which CRU is to be used (primary or redundant).

c. Select which internal nodes are to be used (on-board and off-board).

d. Select the path used to the external nodes (primary or redundant).

e. Assign node addresses.

To be written.

2.6.1 Designating a commander

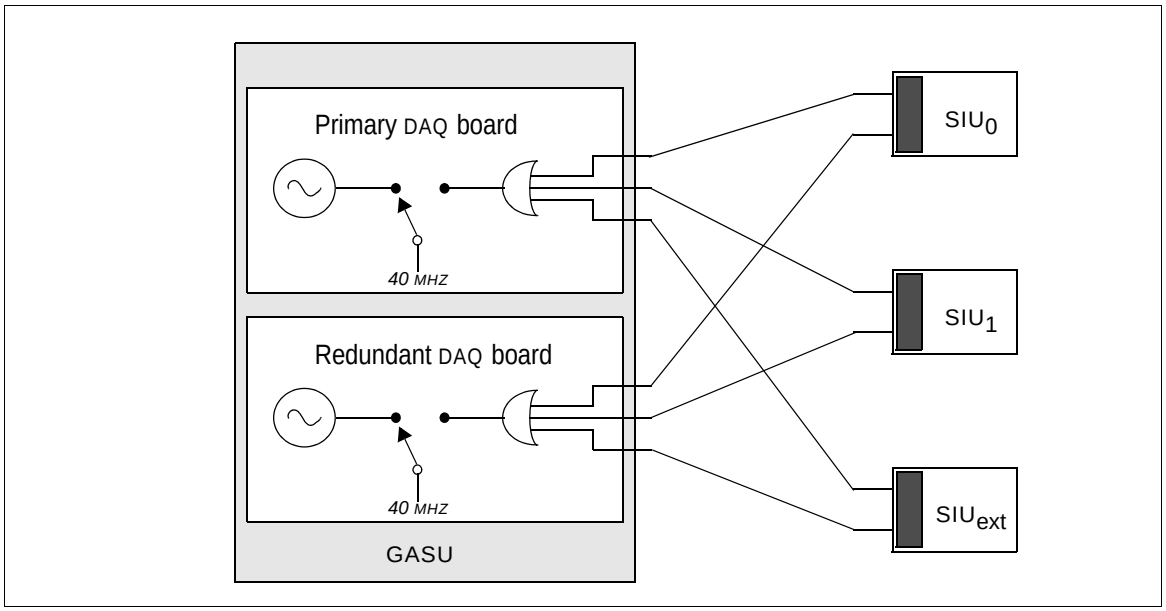

Figure 21 Redundant paths of the Command/Response fabric (external to GASU)

2.6.2 Designating a commander

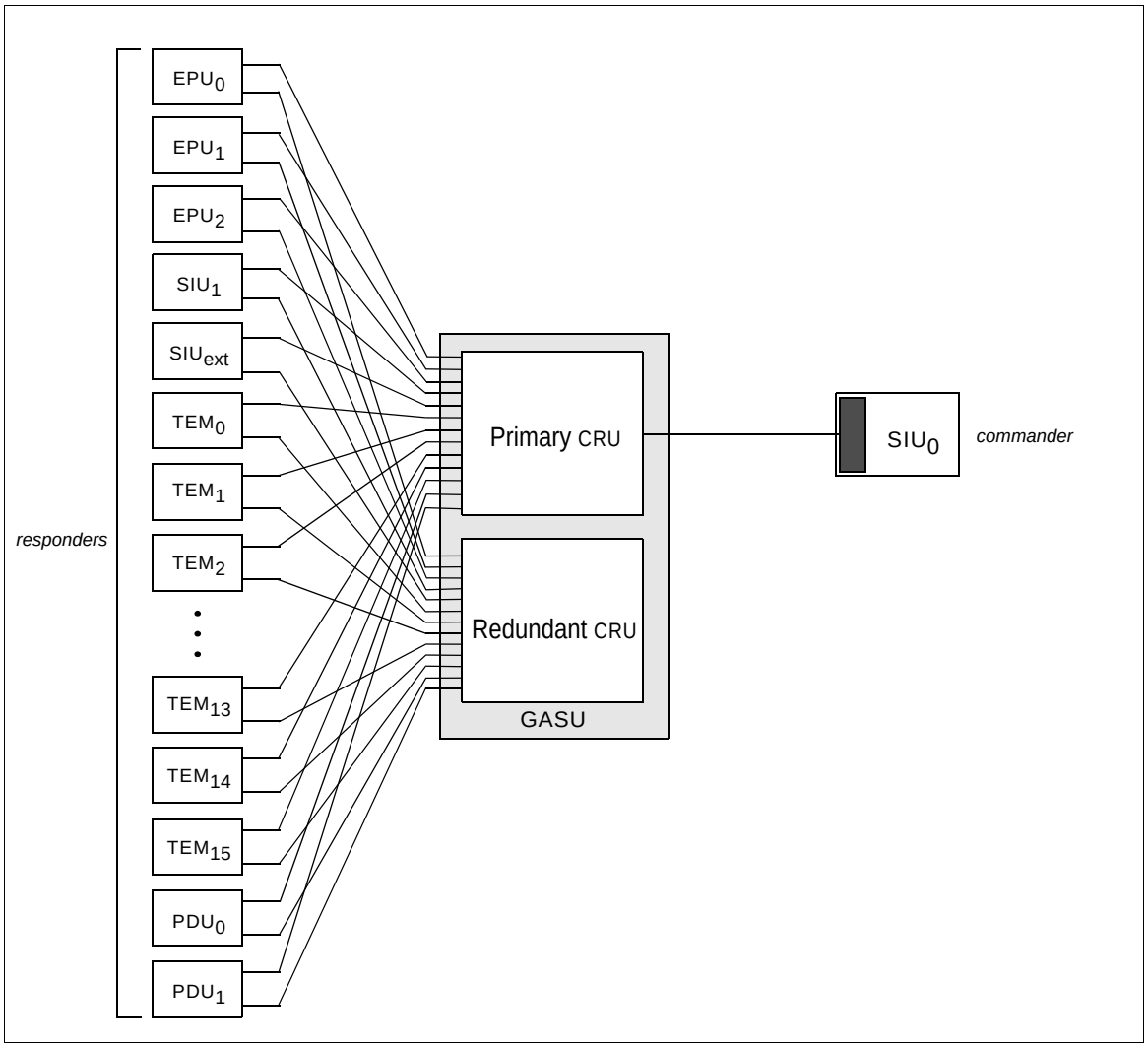

Figure 22 Redundant paths of the Command/Response fabric (external to GASU)

2.6.3 Selecting the internal nodes

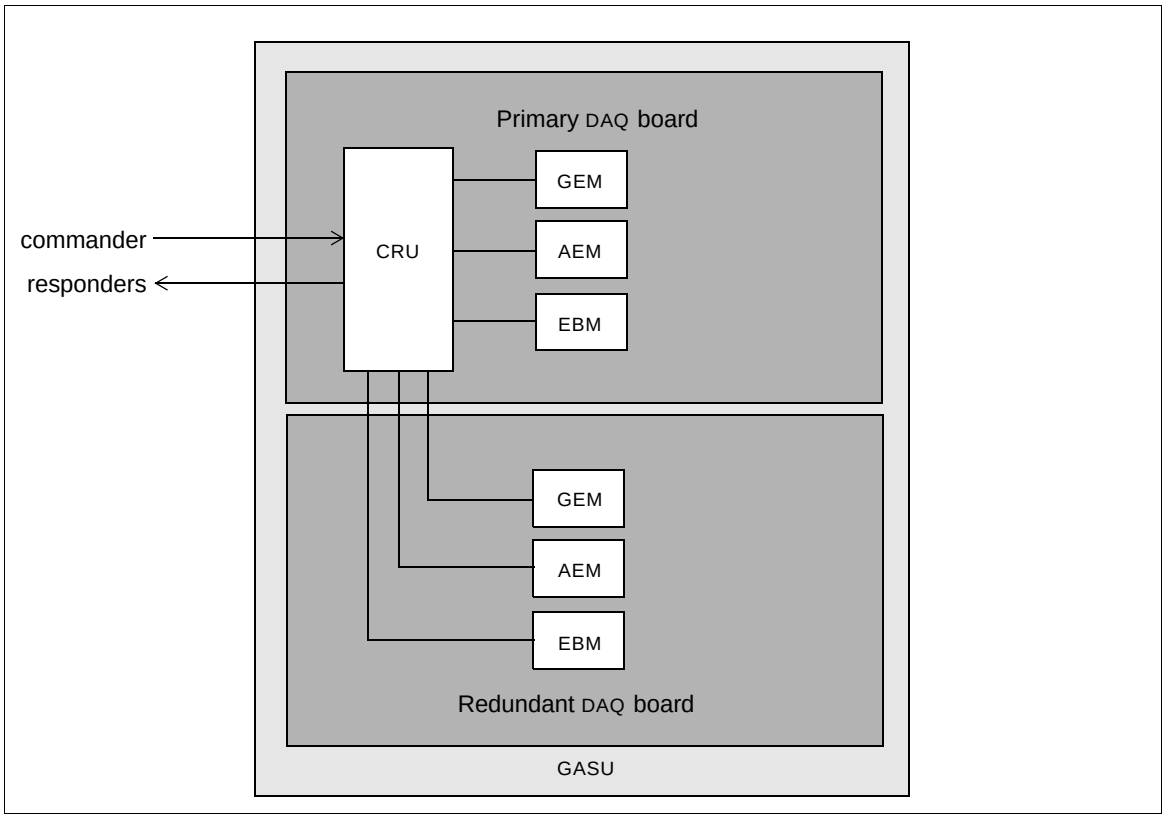

Figure 23 Command/Response fabric after selection of CRU

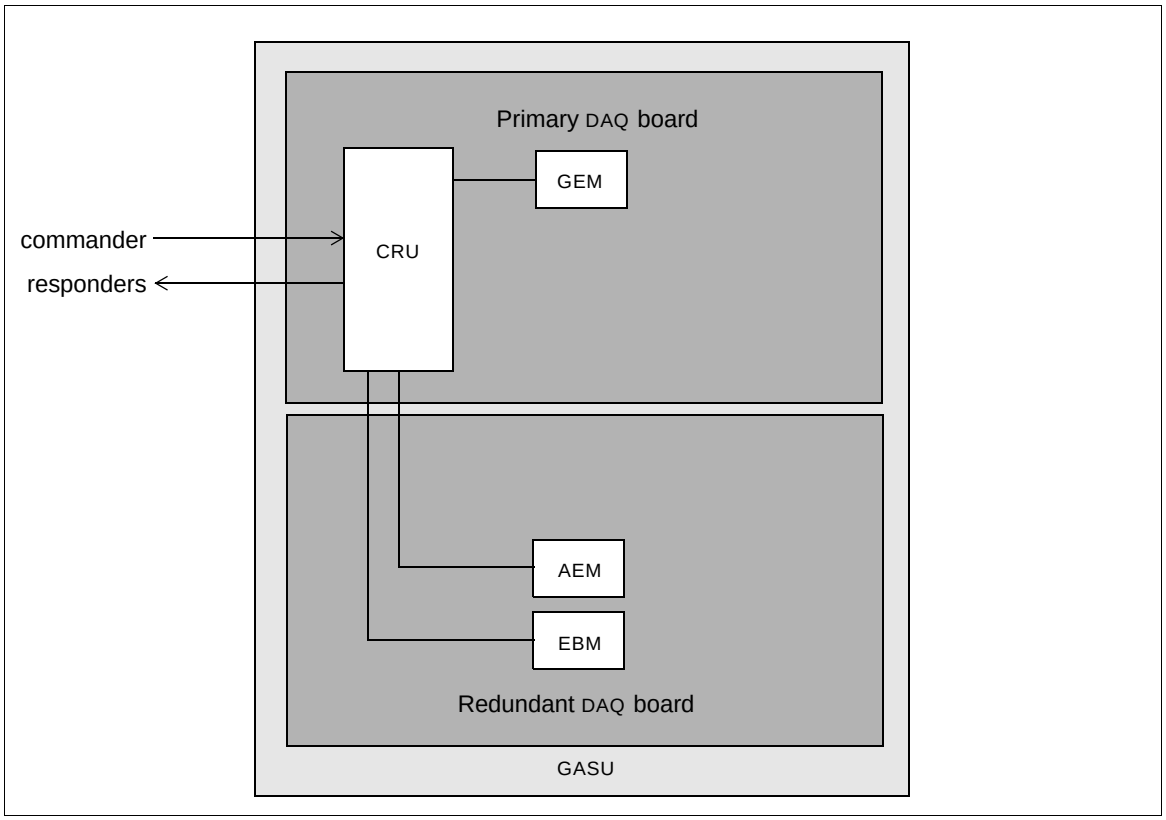

Figure 24 Command/Response fabric after selection of internal responders

2.6.4 Selecting the external nodes

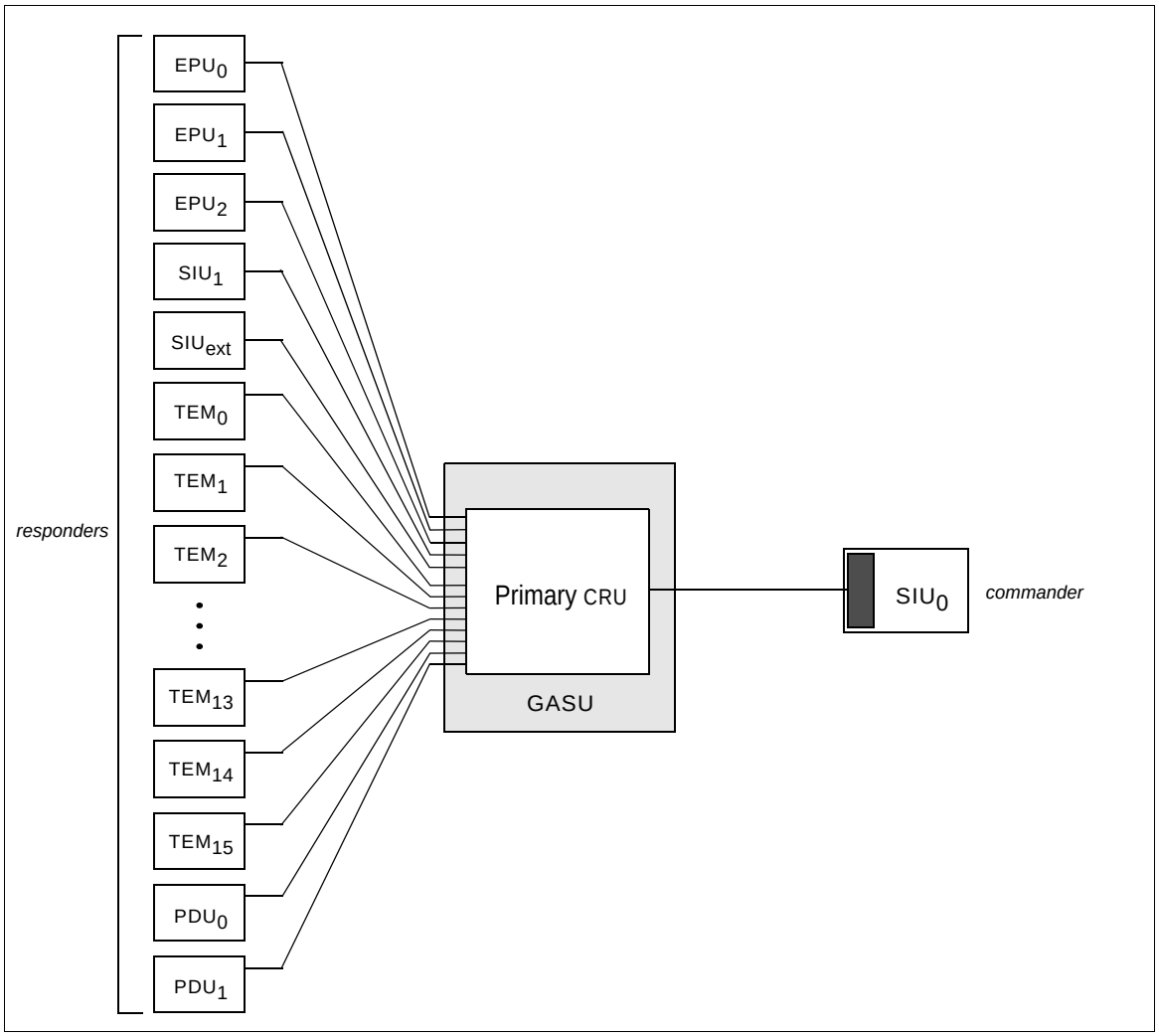

Figure 25 Redundant paths of the Command/Response fabric (external to GASU)

2.6.5 Node Assignments

To be written.

Table 6 Assignment of master node address on the Command/Response fabric

Node	Address[1]	Floats?
EPU_0	00	No
EPU_1	01	No
EPU_2	02	No

Table 6 Assignment of master node address on the Command/Response fabric

Node	Address[1]	Floats?
SIU_{ext}	00	No
SIU_0	01	No
SIU_1	02	NO
Broadcast	1F	No

1. In hexadecimal

Table 7 Assignment of slave node address on the Command/Response fabric

Node	Address[1]	Floats?
Reserved	00	No
TEM_0	01	Yes
TEM_1	02	Yes
TEM_2	03	Yes
TEM_3	04	Yes
TEM_4	05	Yes
TEM_5	06	Yes
TEM_6	07	Yes
TEM_7	08	Yes
TEM_8	09	Yes
TEM_9	0A	Yes
TEM_{10}	0B	Yes
TEM_{11}	0C	Yes
TEM_{12}	0D	Yes
TEM_{13}	0E	Yes
TEM_{14}	0F	Yes
TEM_{15}	10	Yes
GEM	11	Yes
AEM	12	Yes

Table 7 Assignment of slave node address on the Command/Response fabric

Node	Address[1]	Floats?
EBM	13	Yes
PDU_0	14	Yes
PDU_1	15	Yes
CRU	1E	No
Broadcast	1F	No

1. In hexadecimal

2.7 Register model

The protocol's principal function is to provide access to a responder's registers. In order to do so, the protocol adopts the following model of the registers managed by a responder:

- A responder is partitioned into an arbitrary set of different functional blocks. There may be both different *kinds* of blocks and multiple instances of any one type of block.
- Registers are contained within functional blocks. Functional blocks are either *on*-board or *off*-board a responder.
- Accessible registers are contained on either on-board *or* off-board blocks. Registers contained within on-board blocks are ***local***. Registers contained within off-board blocks are ***external***.
- *External* blocks (and their registers) are always accessed through *internal* blocks.
- Commands may be targeted to either a register *or* block.

The register model is illustrated in Figure 26:

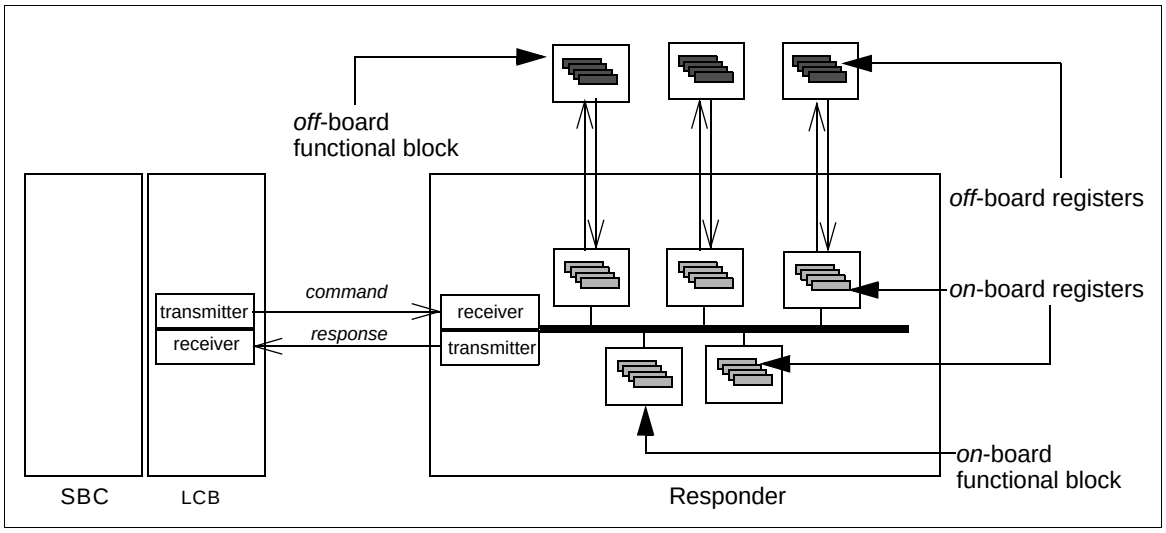

Figure 26 Register Model

2.8 Command string

A ***command string*** is encoded on the data field of the control cell which defines the command packet. The first 9 bits of a command string have a responder independent structure. Following these bits are two generic fields, whose length and structure are responder *dependent*. However, independent of responder, the total length of any one command string may be no more then 112 bits.[1] The form of a command string is illustrated in Figure 27:

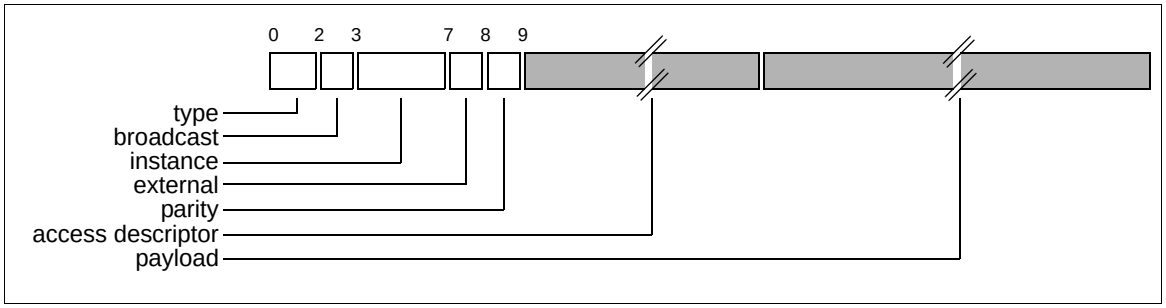

Figure 27 Command *string*

type: A small value enumerating which kind of functional block within the responder will *field* the command. The enumeration values are responder dependent.

broadcast: Specifies whether the command should be *broadcast*. If this bit is *set*, the command is fielded by *all* functional blocks of the specified type.

1. The maximum length of the data field of a control cell.

instance: The responder may contain more then one instance of a particular kind of functional block. This field enumerates which instance of the type (as specified by the previous field) will field the command. The encoding of this field is responder dependent.

external: This bit determines whether the target of a command is either on or off-board the responder. If the bit is *set*, the target of the command is *off*-board. If the bit is *clear*, the target of the command is *on*-board the responder.

parity: The *odd* parity value over the previous 9 bits.

access descriptor Specifies (relative to the functional block specified by the ***type*** and ***instance*** field) both the address of the target and the operation directed at the target. The structure and length of this field are *type* dependent; however, any access descriptor will always contain a two-bit ***function*** field. See Section 2.8.1 for more information.

payload: The parameterization for the command. For example, if the command specifies the writing of a register, the payload would correspond to the value loaded into the register. The width and data representation of the payload depend on the ***access descriptor.*** For example, if the function specifies the reading of a register, *no* payload is required.

2.8.1 The Function field of the access descriptor

Although the structure of an *access descriptor* is responder specific, it must at a minimum contain a two-bit field called the ***function***. This field is simply an enumeration of the possible types of access requested by a command. The enumeration values are found in Table 8:

Table 8 Enumeration of function field

Function	Value[1]
`Dataless`	00
`Load`	01
`Read`	10
Not defined	11

1. In binary

Chapter 3 Events

3.1 Introduction

3.1.1 Generic field definitions

Any field not defined, or used to provide alignment Must Be Zero (MBZ). All undefined fields are illustrated by *graying* out. Any field used as a boolean will have a value of one (1) used to indicate its *set* or *true* sense and a value of zero (0) to indicate its *clear* or *false* sense.

3.2 Event Transport

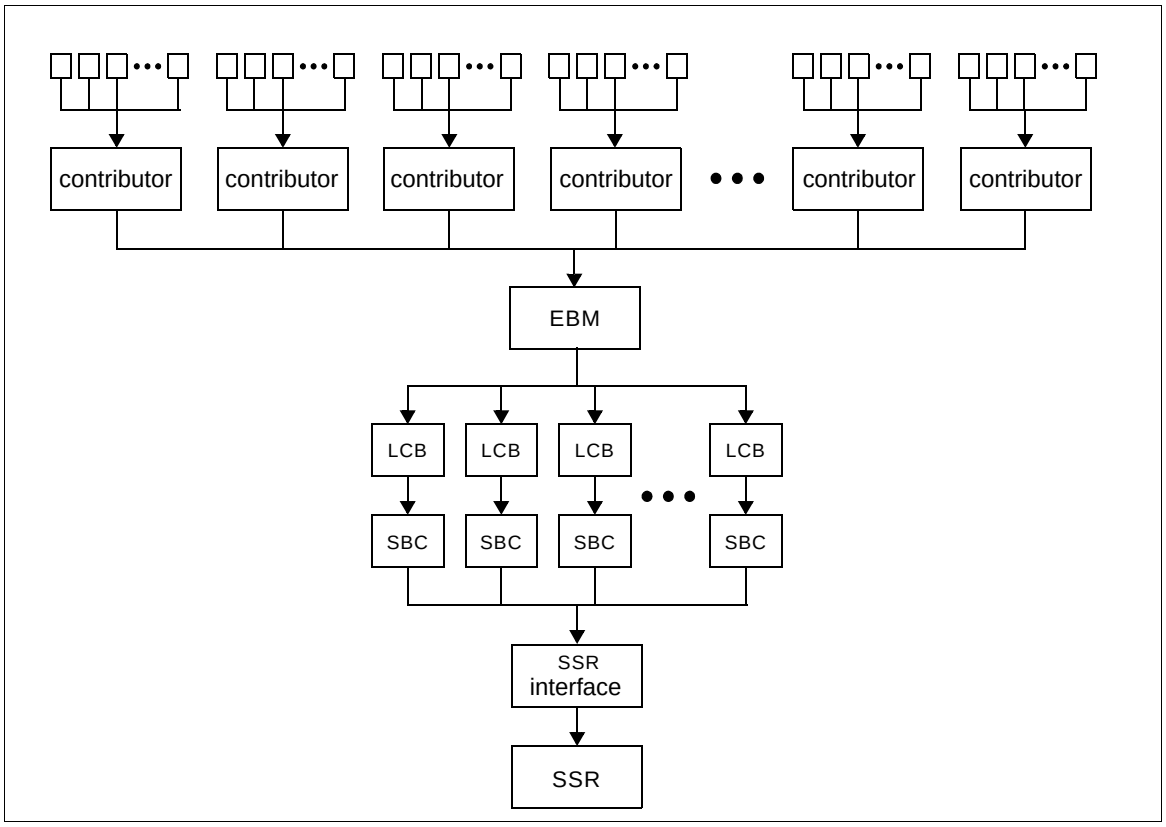

Figure 28 Event Fabric topology

3.2.1 Protocol basis

All contributions are imposed on the cell protocol (LATp) described in Chapter 1.

- One event corresponds to one packet.
- All contributions are rounded *up* to an integral cell size.

The fields of the packet header for a contribution are defined as follows:

- The ***destination address*** is derived from the *trigger message*.
- The ***respond*** field is *false*.
- The ***source address*** is the source's physical address.
- The ***protocol*** value is set to *zero* (0).

3.3 Event shape

The information in this section is obsolete and has been supersede by information found in [4].

3.4 Contribution shape

Any contribution may consist of as few as two and as many as four different *kinds* of data:

summary: A fixed 32-bit structure summarizing the event. Most of the fields of the ***summary*** are derived from the incoming trigger message corresponding to the emitted event.

event: The normal data contribution from the source's Front-End Electronics.

diagnostic: An arbitrary amount of supplemental data appended to a source's normal event contribution. It is intended to be generated only while either performing in-flight diagnostics or commissioning electronics. Its purpose is to resolve conflicts in the accuracy and veracity of the data *output* from a source, as opposed to the data *input* to a source. For example, diagnostic data for the calorimeter may include its log accept bits, in order to validate the TEM's zero-suppression machinery.

error: An arbitrary amount of data appended to a source's normal event contribution, whenever an error is detected in transporting or constructing the normal data contribution. This data will characterize the nature and location of an error.

Both the size and composition of a source's contribution vary as a function of which of these types are present in a contribution. *Which* types are present is determined by the values of the ***error*** and ***diagnostic*** fields of the ***Summary***. (See Section 3.5.) However, there are certain invariants independent of type composition:

- In order to have data begin on a word (32-bit) boundary with respect to the packet header, a half-word (16-bits) of padding always immediately follows the packet header.
- The ***Summary*** is *always* present and follows the prefix padding.
- In order to round contributions up to cell boundaries, a variable amount of padding is necessary between the end of the data and the end of the cell. As all data is guaranteed to be a multiple number of words in length, this padding may vary from zero to three words.

While the ***error*** and ***diagnostic*** fields imply at least four different combinations, it is perhaps clearer to consider these fields as specifying two different *shapes*, which may or may not be in *error*. Thus a contribution could have its *natural* shape (containing a summary and event data) or its *diagnostic* shape (containing a summary, event and diagnostic data), and either shape could be in error. The invariants and the two different shapes are illustrated in Figure 29 and Figure 30.

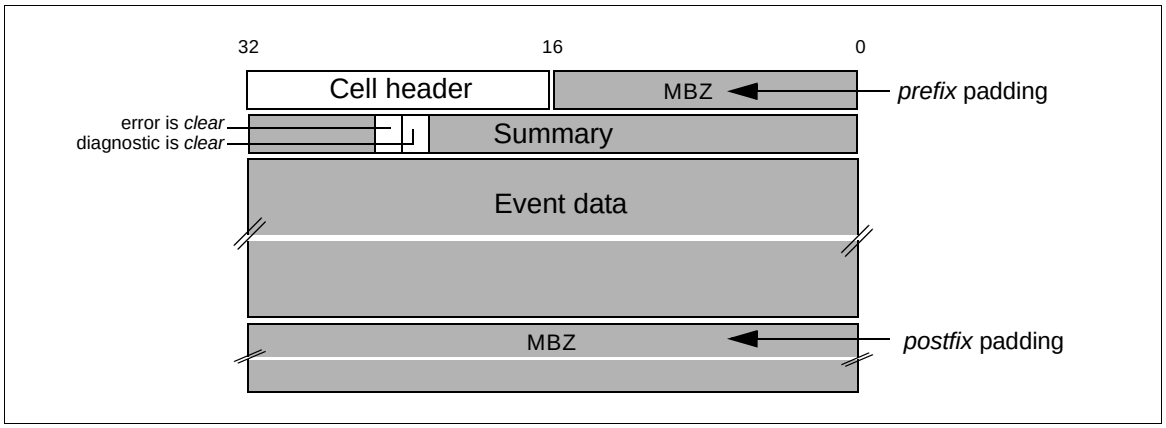

Figure 29 *Natural* contribution

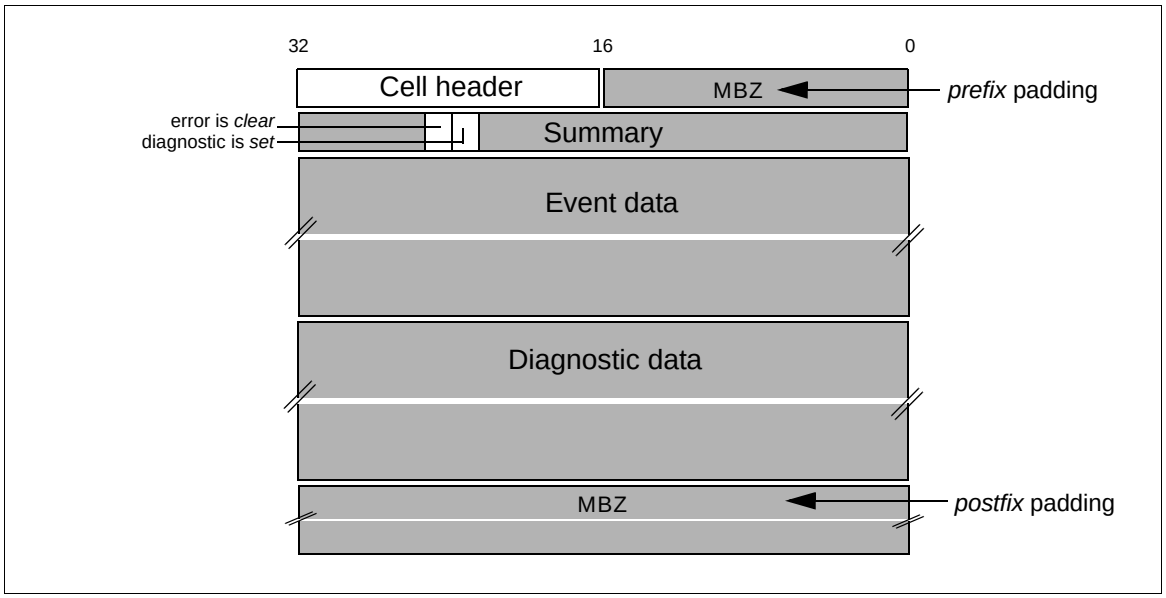

Figure 30 *Diagnostic* contribution

Either of these two shapes could be in error. Therefore, there are two additional combinations which depend on the state of the ***error*** bit. These are illustrated in Figure 31 and Figure 32.

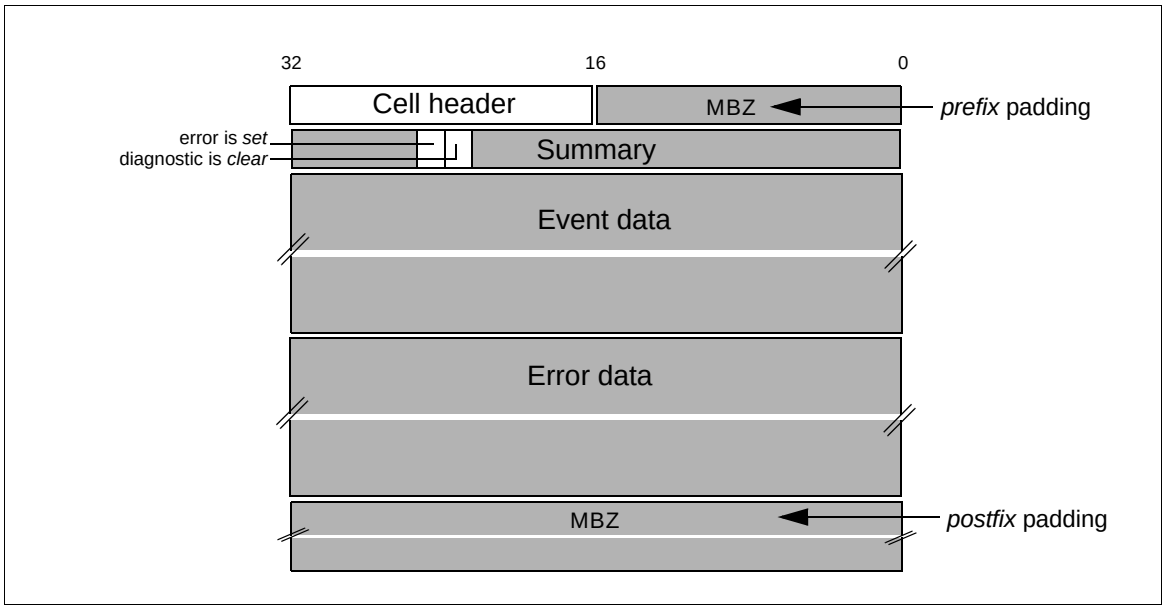

Figure 31 Natural contribution in error

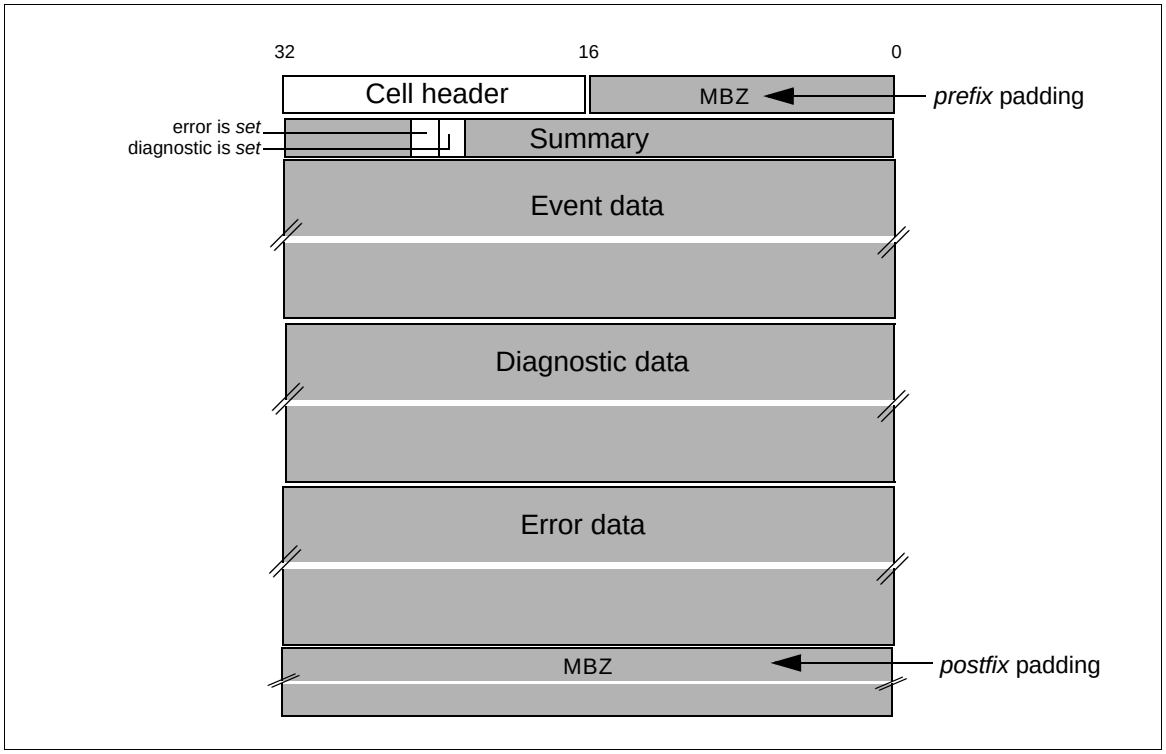

Figure 32 Diagnostic contribution in error

3.5 The Event Summary

The first significant word of any contribution is the ***Event Summary.*** Many of the fields are simply a copy of the analogous fields of the trigger message which caused the event to be emitted. (See Section 4.) The structure of the ***Summary*** is illustrated in Figure 33:

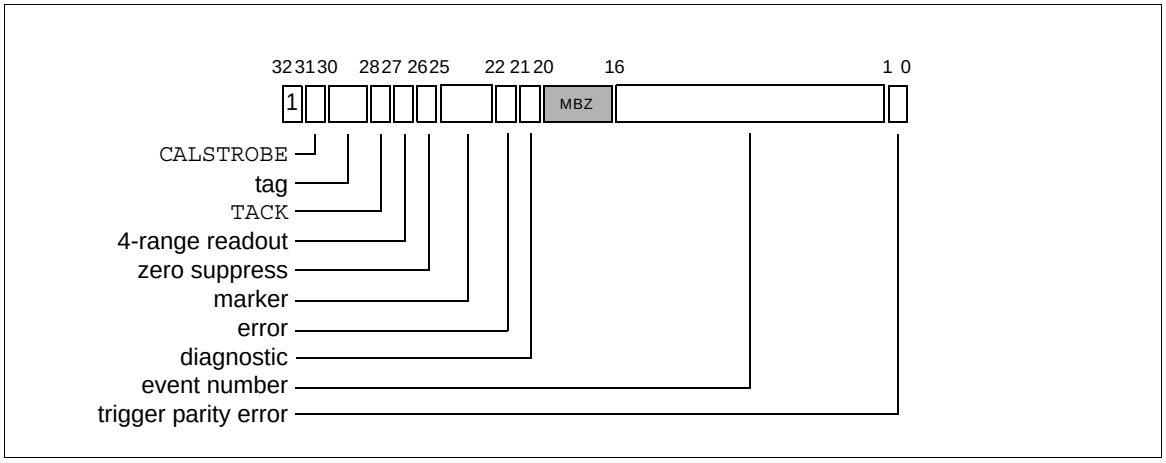

Figure 33 Structure of the Event summary

CALSTROBE: Specifies what type of command was *first* requested by the GEM. If this field is true (*set*), the first command requested was a CALSTROBE. If the field is false (*clear*), the first command requested was a TACK. (See also the ***TACK*** field.) This field is a copy of the analogous field of the *trigger message*.

tag: The two *least* significant bits of the 17-bit event sequence number. The remaining 15 bits of the sequence number are contained in the ***event number*** field. This field is a copy of the analogous field of the *trigger message*.

TACK: Specifies whether a *second* command was requested by the GEM. If this field is true (*set*), a second command was requested (following the first command specified in the ***CALSTROBE*** field). This second command is always a TACK. If the field is false (*clear*), a second command was *not* requested. This field is a copy of the analogous field of the *trigger message*.

4-range readout: If this field is *set*, the Front-End Electronics emitted all four ranges of its event data. If *clear*, the data was auto-ranged and the one appropriate range for its event data was sent. This field is a copy of the analogous field of the *trigger message*.

zero suppress: If this field is *set*, the source was requested by the GEM to zero suppress any event data emitted. If this field is *clear*, the source did *not* zero suppress event data. This field is a copy of the analogous field of the *trigger message*.

marker: Used by flight software in order to insert markers at well-known times into the event stream. This field is a copy of the analogous field of the *trigger message*.

event number: The 15 most significant bits of the 17-bit sequence number. The low-order two bits of the sequence number are contained in the ***tag*** field. This field is a copy of the analogous field in the *trigger message*.

error: If this field is *set*, the contribution obeys the format of either Figure 31 or Figure 32. If the field is *clear*, the format obeys either Figure 29 or Figure 30. See Section 3.6 for more information.

diagnostic: If this field is *set*, the contribution obeys the format of either Figure 30 or Figure 32. If the field is *clear*, the format obeys either Figure 29 or Figure 31. See Section 3.4 for more information.

trigger message parity error: Specifies whether the computed parity of the trigger message which caused the event to be emitted was incorrect. If the field is *set*, a parity error occurred.

3.6 Error processing

During in the construction of an event a source may discover an inconsistency in the data it is building. When there is such an occurrence, the constructed event is said to be in error. A source has three responsibilities when an error occurs:

i. *Sets* the ***error*** field in the ***Summary*** word of the event contribution.

ii. *Latches* the ***error*** field in a status register.

iii. *Appends* to the event's natural shape an appropriate error description.

Chapter 4 The Trigger

This chapter is obsolete. Its information has been superseded by information found in [5].