The GLAST experiment at SLAC

The GLT Electronics Module

Electronics group

Programming ICD specification

Document Version:	2.7
Document Issue:	1
Document Edition:	English
Document Status:	Under release control
Document ID:	LAT-TD-01545-02
Document Date:	March 16, 2005

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

A conceptual design of the GEM (GLT Electronics Module) and the role it plays within the LAT Trigger System is described. The GEM is the electronic realization of the Global (GLT) Trigger. Its programmatic interface, used to both configure and monitor its behaviour, is defined and described. Last, the contribution of the GEM to the information contained within an event is defined and described.

Hardware compatibility

This document assumes the following hardware revisions:

TEM: Version TBD

Intended audience

This document is intended principally as a guide for the *developer* and *users* of the GLT Electronics Module (GEM). Users include:

- Developers of the sub-system electronics which interface with the GEM
- 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] and indeed, most the references described within.

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 "LAT Inter-module Communications - A reference manual," Michael Huffer, LAT-TD-00606.

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

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

4 *Actel* data sheet for RadTolerant FPGAs for Space Applications (RT54SX-S).

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

6 "Tower Electronics Module - Programming ICD specification," Michael Huffer, LAT-TD-00605.

7 "Specification of the Glast Tracker Readout Controller Electronics (GTRC)," Jeff Olsen, LAT-SS-00170.

8 "Large Area Telescope (LAT) Instrument-Spacecraft Interface Requirements," 433-IRD-0001.

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:**	The GLT Electronics Module Programming ICD specification		
	Version:	2.7		
	Issue:	1		
	Edition:	English		
	ID:	LAT-TD-01545-02		
	Status:	Under release control		
	Created:	February 9, 2002		
	Date:	March 16, 2005		
	Access:	V:\GLAST\Electronics\Design Documents\GEM\v2.7\Frontmatter.fm		
	Keywords:	GLT Electronics Module GEM		
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		

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:	The GLT Electronics Module Programming ICD specification		
ID:	LAT-TD-01545-02		
Version	**Issue**	**Date**	**Reason for change**
1.0	1	5/29/2002	Initial draft
1.1	1	6/25/2002	Incorporated comments from Curt, Gunther, JJ, and Tony
1.2	1	1/07/2003	Major changes, moving ever onwards towards a *real* document. This release is only so JJ, Tony, and Jozesf can have a look- see.
1.3	1	2/17/2003	Still moving onwards... I now believe that chapters 2, 3, and 4 are substantially correct. This release is only so JJ, Tony, James, and Jozesf can have a look-see.
1.4	1	2/20/2003	Mostly, fixing typos as result of comments by James and JJ. Changed measurement of deadtime to measurement of live-time. Increased time-base counter to 25 bits and decreased 1-PPS counter to 7 bits. Increased rate of periodic trigger to 160 KHZ.
1.5	1	4/22/2003	Effectively, the entire document has been re-written. Lots of detail added on the Trigger *System* and role the GEM has within this system. However, the GEM itself, has not changed that much. Here are the highlights: — re-defined window width definition — added version identification (configuration register) — re-thought and completely revamped whole idea of “bound” and “free-run modes”. Now moved to control over periodic trigger. — periodic trigger rates modified (and explained) — added *inhibit* field to Message engine. Changed definition of pre-scalers.
1.6	1	5/02/2003	Mostly, fixing typos as result of comments by James and JJ. Changed measurement of deadtime to measurement of live-time. Increased time-base counter to 25 bits and decreased 1-PPS counter to 7 bits. Increased rate of periodic trigger to 160 KHZ.

Table 3 Document Status Sheet

1.7	1	9/22/2003	Changes to reflect the reality of actually building the “beast”. The important changes are in the register interface. They include: — re-ordering of the registers in the controller — added new register block (the `WINDOW` block), moved `WINDOW_OPEN_MASK` register and `WINDOW_WIDTH` fields from `CONFIGURATION` register to this block. — added new fields to `CONFIGURATION` register. — completely redefined meaning of fields of registers within the `INPUT_ENABLE` block.
1.8	1	2/10/2004	Changes to reflect the reality of actually building the “beast”. The important changes are in the register interface. They include: — re-ordering of the registers in the controller — added new register block (the `WINDOW` block), moved `WINDOW_OPEN_MASK` register and `WINDOW_WIDTH` fields from `CONFIGURATION` register to this block. — added new fields to `CONFIGURATION` register. — completely redefined meaning of fields of registers within the `INPUT_ENABLE` block.

Table 3 Document Status Sheet

2.0	1	3/08/2004	Many, many, changes to reflect "as built". These include (in no particular order): — Incorporated GXH's comments (of a year ago!) — *All* registers are now 32-bits — Configuration register completely different. More control over "dummying" up parity. Removed mixing and matching of modules, as this is no longer supported by the GASU. New version information, to reflect the fact that the GEM has three FPGAs, each with their own lifetime. — Added field in Configuration register to inhibit production of GEM's event contribution. — Made connection between ACD naming conventions and FREE board, tile number etc. See new appendix. — Cleaned up notation. Trigger Summary is now *Condition* Summary. Template-X are now *Engine*-X. Indices-x are now *Conditions*-X. — Eliminated so-called "trigger order" in ROI and enable definition registers. Registers are now arranged in FREE board order. — Entire handling of statistics counter(s) has completely changed. — New counters for counting singles rates on ACD tiles. — Periodic trigger rate register is now specified as a prescale, rather then an exponent. — Renamed "busy" counter to "discarded" counter. — Inverted "busy" counter and "prescaled" counter in event data. — Changed default sense of "periodic mode register" in order to have periodic condition come up disabled by default. — Window width register used to have built-in offset of four (4) clock tics. This is no longer true. Beware: Do not program this register with a value of *zero*. — Added one more piece of information to the GEM's event contribution. This piece of information computes the *delta* time between events. — By mistake, exposed high order bit of destination address, used in configured engines. This field is always an implied one (1), as events can only be sent to a master on the event fabric. Field is now *Read-Only*. Updated fonts.

Table 3 Document Status Sheet

2.1	1	4/22/2004	Minor changes. Mostly reflect fixing typos and the fact that the structure of the configuration register changed and a documentation of the tile mapping for the “first article”. The changes include: — structure of CONFIGUARTION register changed (removed 1-PPS selection). — `PERIODIC_TRIGGER_RATE` register changed. Counter is actually 24 bits rather then 16 bits. — Mapping of ROI generator (table 15) was documented incorrectly. Middle entries were shifted by one. — Tile mapping for the “goofy” board (first ship) does not match documentation. This is fixed on the next layout as it required to many new wires on the current board (see appendix A).
2.1	2	4/23/2004	Corrected some typos.
2.2	1	4/29/2004	This version represents the changes in going from the “first” generation GEM to the “second” generation GEM. There are three: First, as the new board has one more FPGA, the version structure has changed in the configuration register. Also, as this is a new board, the board revision number has been incremented, which will allow one to differentiate old boards from new boards. Second, exchanged the order of the ribbon tiles (500x and 600x) to make the tile mapping monotonically increasing. This changed both the ROI and tile enable registers (and appendix A). Third, the tower “busy” enables were moved to their own register (register 17) rather then sharing
2.2	3	5/18/2004	Corrected some typos.
2.2	5	6/10/04	Updated references and PDF TOC.

Table 3 Document Status Sheet

2.3	1	9/01/04	Updated to reflect new hardware functionality Added new CNO counters. Added new condition: the external trigger.
2.4	1	10/18/04	Some new functionality and re-definition of the tracker *not* vetoed condition (which has always been wrong). These changes are reflected in a hardware version number of five (5) for both the scheduler and TAM. The ROI version is "don't care". The changes include: — Added access to 1-PPS counter and register (see Section 2.6.4) — Reused the sent event contribution and replaced this contribution with a measurement of the relative arrival times of the conditions present in the event. (see Section 4.13). Note: this change is *not* back-ward compatible. — Added enable for external trigger condition. This feature was omitted when I added the external condition. Note: As the external condition, was by default previously *enabled* and is now by default *disabled*, this change is *not* backwardly compatible. (see Section 2.6.3) — The tracker *not*-vetoed definition was incorrect. Nothing is changed in the hardware, however, the condition has been renamed to tracker vetoed. (see Section 1.7.1 and Figure 30) — Added on more bit of range to livetime counter (now 25 bits)
2.5	1	01/26/05	The changes are: — Added access to the GEM's internal time-base (see Section 2.6.5) — For some, unidentified reason, I never updated the description of the Periodic Trigger Rate Register (see Section 2.3.3.1) and Periodic Limit Register. (see Section 2.3.3.3). Don't know why - its been that way in the hardware forever. Thank's Gregg for pointing this out.
2.6	1	03/01/05	The changes are: — Added new register to count potential window turns in "dead zone" (see Section 2.6.1.2) — Least significant 8 bits of dead-zone counter are sampled and returned in event contribution (see Section 4.8). — Added load descriptor for statistics block (was implemented in hardware and software, but never documented). See Section 3.5. — Removed dataless command descriptor for statistics block (was never implemented in hardware or software, and is no longer required). See Section 3.5. — Corrected (some) typos and added a little more explanation of a window (but still needs more).

Table 3 Document Status Sheet

2.6	2	03/05/05	Updated explanation of how statistics are actually reset (see Section 2.6.1).
2.7	1	03/16/05	Added a new register in order to allow programmatic control of the external trigger condition (see Section 2.3.7). Corrected some typos and added additional clarification of the dead-zone (see Section 2.6.1.2).

Table of Contents

List of Figures

List of Tables

Chapter 1
Principles of operation

1.1 Overview

The principal function of the GEM (GLT Electronics Module) is to process *Trigger Inputs* from the LAT's ACD and sixteen towers and from these signals arrive at a decision on whether or not to *trigger* a readout of detector data. The sum of all the information read out of the detector electronics when triggered is called an *event*.

The ACD's principal function is to reject background in the towers. Thus, from the perspective of the GEM, the primary purpose of the inputs provided by the ACD is as a trigger *veto*. However, to assist in absolute energy calibrations, a second set of signals is developed by the ACD, to be used by the GEM as a *trigger*. Thus, inputs from the ACD fall into two categories:

Vetos: A discriminated signal from each of the 97 tiles of the ACD, used to (potentially) veto tracker triggers originating in any one of the sixteen towers.

CNO: A discriminated and summed signal representing highly ionizing particles from heavy nuclei (Carbon-Nitrogen-Oxygen). This input is used as a trigger.

The sixteen towers are identical in composition and each one consists of both tracker and calorimeter systems. Thus, inputs from the towers fall within three categories:

TKR: A three-fold coincidence between any three adjacent *Tracker* layer pairs.

CAL (LE): Deposited energy anywhere within the *Calorimeter* higher than a discriminated value. The calorimeter produces *two* discriminated values: *Low* and *High* energy. This is the *Low* Energy Trigger Input and is referred to as $\mathrm{CAL_{LE}}$.

CAL (HE): Deposited energy anywhere within the *Calorimeter* higher than a discriminated value. The calorimeter produces *two* discriminated values: *Low* and *High* energy. This is the *High* Energy Trigger Input and is referred to as $\mathrm{CAL_{HE}}$.

And finally, those inputs which are developed in order to both monitor and debug the triggering system itself:

Periodic: A periodic, fixed rate signal. The frequency of this signal is derived from either the system clock or the 1-PPS signal.

Solicited: This signal is asserted through the Command/Response interface, allowing the system to be triggered through operator intervention.

When suitably consolidated by the GEM, these seven different categories also define *trigger conditions*. Each condition corresponds to one reason why the detector should be read out. Conditions are used by the GEM for two different, but equally important functions:

- In order to *open* the *trigger window*. The trigger window defines a period in time in which inputs are defined to be in coincidence. The *width* of this window is a configurable parameter of the GEM. That is, a window is *closed* a fixed time after it is opened. Each time a window is opened and then closed is called a window *turn*.
- To determine whether a triggerable *condition* exists in the detector. It's possible, even likely, that more then one condition could be present during the time a window is open. The set of conditions which have occurred between a window opening and closing are a window's *Condition Summary*. As there are eight potential conditions, each window turn could create one of 2^8 (256) possible condition summaries.

Only non-zero condition summaries are candidates for triggering the system. However, a non-zero summary is, in and of itself, not sufficient to guarantee a positive trigger decision and corresponding readout of the LAT. In order to actually trigger a detector readout, a non-zero summary on a window turn must satisfy two additional constraints:

- The *prescaler* associated with the summary must be expired.
- The detector electronics involved in event readout are not *busy*.

Prescaling allows the user to specify what fraction of the natural rate of the conditions will be passed through the readout of the system. The GEM contains *sixteen* prescalers. The GEM allows its users to associate any of the 127 possible trigger conditions with any one of its sixteen prescalers through the configuration of the *Message Engine lookup table*. The association between Condition Summary and prescale is discussed in 1.7.1.

The readout electronics are themselves heavily buffered, and both event readout and event building occur asynchronously with respect to trigger decisions. Because of this, it is entirely possible that a triggerable condition (or conditions) could occur at a time in which the readout electronics would not have the capacity to either absorb or forward the resulting event. In order to deal with this situation, the LAT electronics architecture specifies a flow-control model involving *back-pressure* coupled with a trigger *throttle*. The signal asserted by the readout electronics to the GEM in order to throttle the trigger is called the *busy* signal. A busy signal is asserted to the GEM from both the electronics servicing the sixteen towers (the TEM, or Tower Electronics Module) and the electronics servicing the ACD (the AEM, or ACD Electronics Module).

Once the GEM arrives at a decision to read out the LAT, it must communicate this decision to all modules responsible for event readout. It is important to note that it is not just the decision which is broadcast to the modules. In addition, the modules also receive from the GEM, a fixed amount of information which the GEM associates with the decision. This information is called the *Trigger Accept Message*, or TAM. Recipients of this message use its contents to perform destination-specific readout of their respective systems. A discussion on how the GEM determines the contents of a TAM is found in sections 1.8.1 and 1.8.4.

A block diagram of the GEM, its input and output interfaces is illustrated in Figure 1:

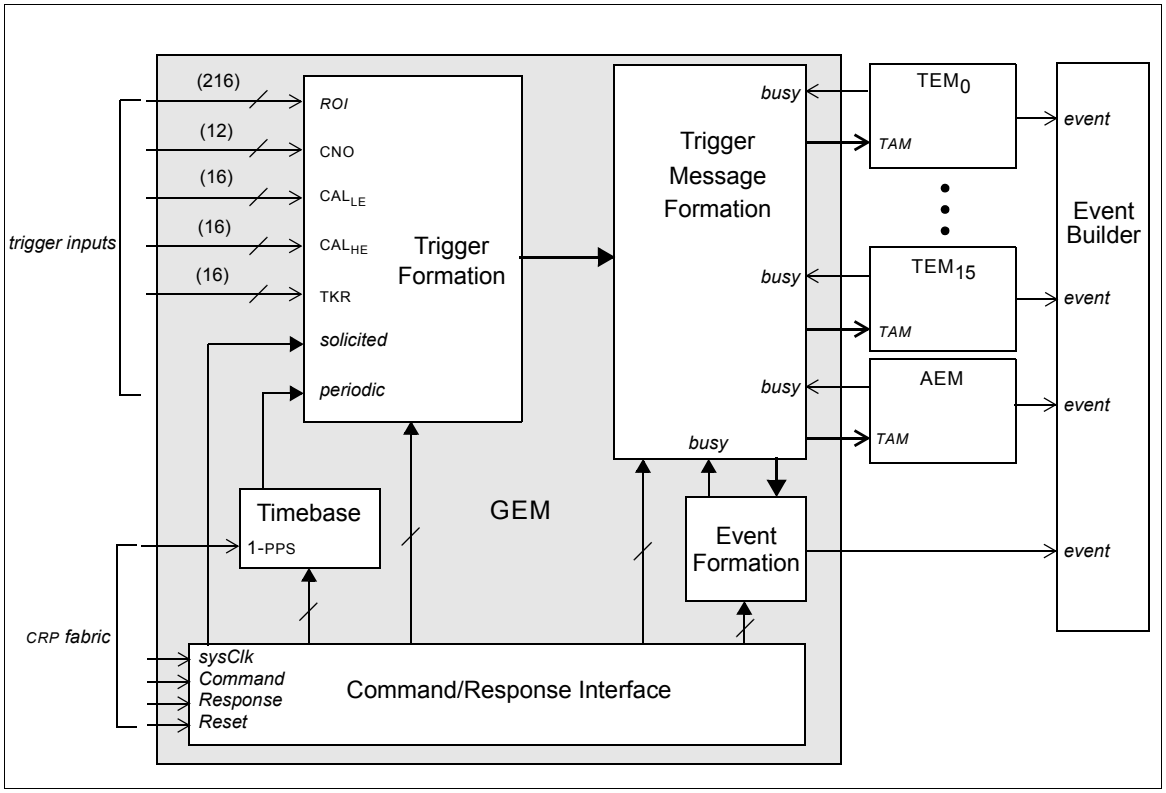

Figure 1 The GEM and its input and output interfaces

In this diagram, trigger inputs arrive from the left and are used to both open windows and formulate trigger conditions. A timebase based on the 20 MHZ system clock and 1-PPS signal is used to synthesise the periodic trigger input signal. These candidate triggers are passed to the message formation block, which prescales the candidates and using busy signals from both the TEMs and AEM, determines whether the LAT should be actually triggered and read out. If so, a TAM is broadcast to the TEMs and AEM. These modules will use the message's contents to perform sub-system specific readout of their respective detectors. Eventually this data will be transferred from the module to the EBM (Event Builder Module), where event contributions will be built and subsequently transferred to the L3 filter farm. (See [2].)

Note: The event data path downstream of the GEM is heavily buffered. Consequently, while not readily apparent in the diagram, the transfer of events occurs asynchronously with respect to operation of the GEM.

The GEM contains two other blocks of interest:

Event Formation: The GEM itself contributes data to the event. The format and structure of this contribution is described in Chapter 4.

Command/Response Interface: The GEM contains a variety of registers. (See Chapter 2.) These registers have two functions: First, they are used to define and specify the configuration of the GEM. Second, they maintain statistics on its operation. In order to access these registers, the GEM appears as a node on the Command/Response fabric. (See [3].) The specific encoding of command and response packets is described in Chapter 3.

1.2 Physical partitioning of the GEM

The combinatorial logic necessary to implement the logical blocks described in the previous section (and shown in Figure 1) is distributed over three ACTEL RT54SX series FPGAs. (See [4].) They are named as follows:

- The ROI (Region of Interest) generator
- The Scheduler
- The TAM (Trigger Accept Message) generator

Together, the ROI generator and Scheduler are responsible for trigger formation. The Scheduler also contains the timebase. The TAM generator is responsible for TAM and Event (contribution) formation as well as implementing the Command/Response Interface. In addition to its three FPGAs, the GEM contains 18 copies of a single custom ASIC called the GLTC (GLAST LVDS Translator Chip). This ASIC is used to condition and process LVDS trigger input signals. Finally, there are a number of commercial LVDS transmitters/receivers used to process miscellaneous control and output signals. For pedagogical reasons, these components, their inter-connections, and external I/Os are illustrated through three different diagrams. (See Figures 2, 3, and 4.) Each diagram represents a somewhat disjoint activity on the part of the GEM. For example, Figure 2 illustrates the connectivity to support the pure triggering activity on the part of the GEM, Figure 3 illustrates the connectivity to support generating the event contribution of the GEM, and Figure 4 illustrates the Command/Response Interface to the GEM's registers. By "overlapping" the three diagrams, the totality of the physical implementation of the GEM can be imagined.

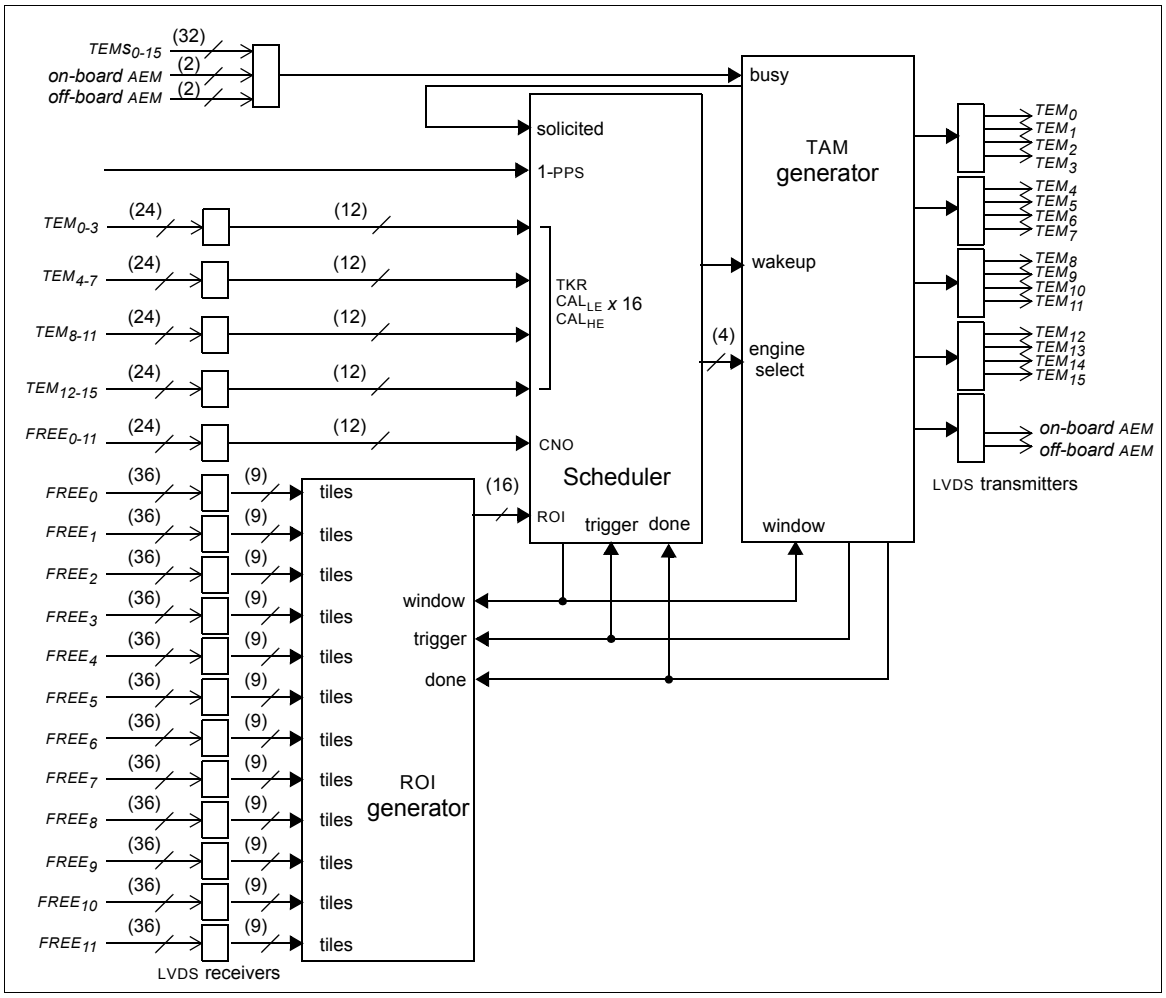

Figure 2 Block diagram of the GEM and its trigger paths

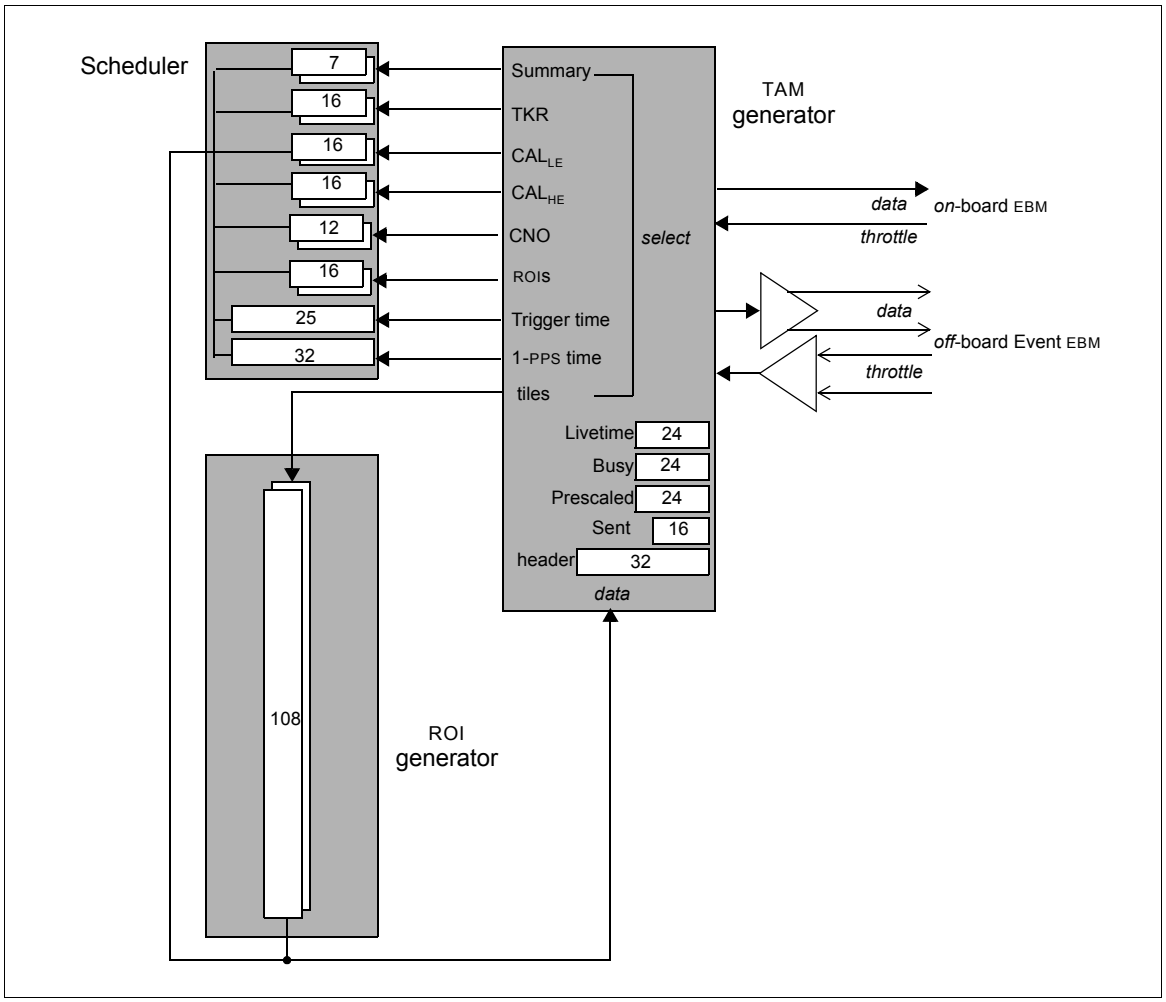

Figure 3 Block diagram of the GEM and its event paths

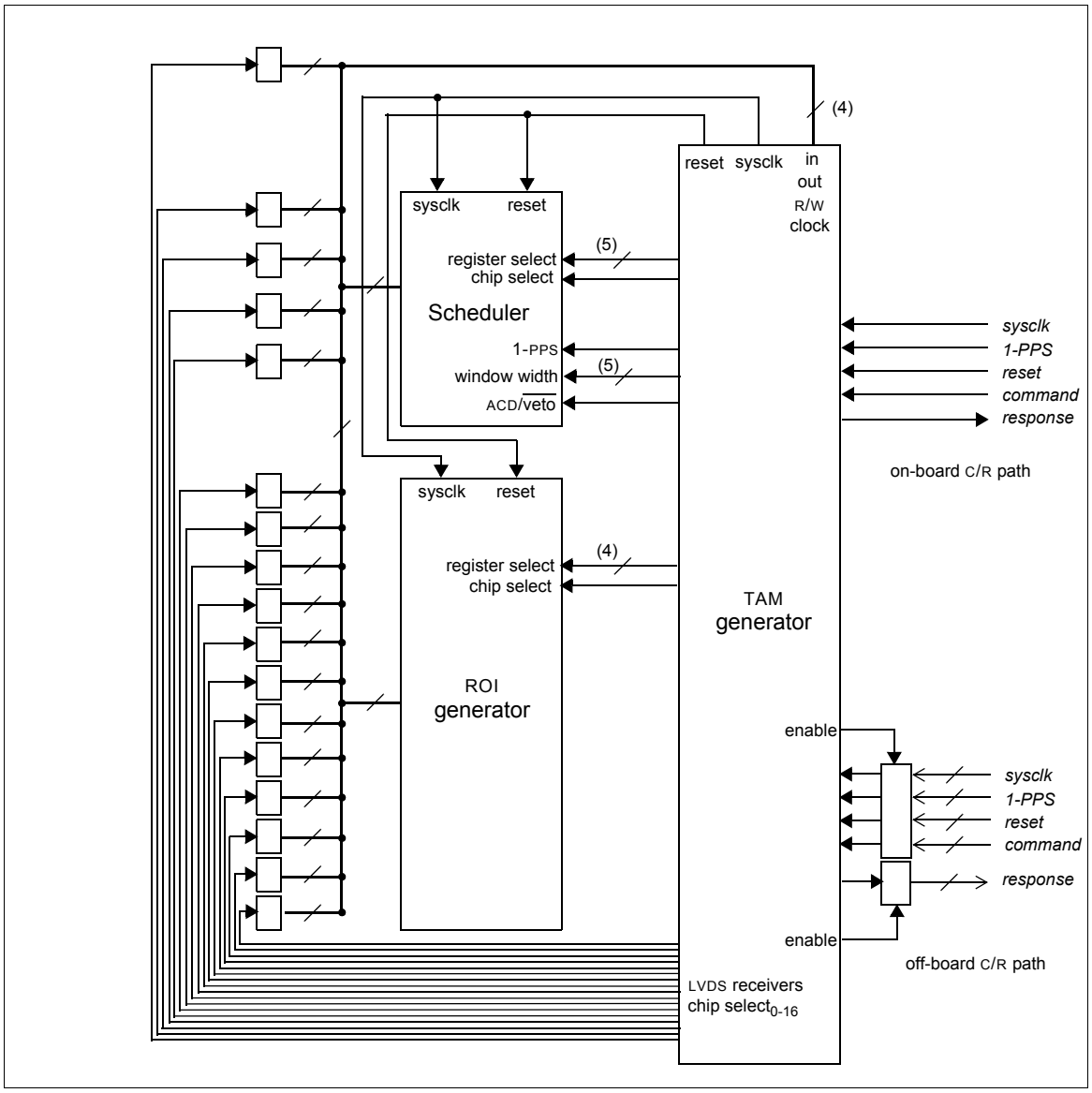

Figure 4 Block diagram of the GEM and its Command/Response paths

The components of the GEM are organized along with the AEM, EBM, and CRU (Command/Response Unit) on a single Printed Circuit Board (PCB) called the DAQ board. The DAQ board resides in the GASU *box* as described in [5]. 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. The GEM on the primary DAQ board is called the *primary* GEM and the GEM on the redundant DAQ board, the *redundant* GEM. In typical operation, only one of two DAQ boards is powered; consequently, only one of two GEMs can operate at any one time. However, the operating GEM can be either the primary or redundant module.

1.3 Trigger Sources

1.3.1 The ACD

The ACD detects energetic cosmic ray electrons and nuclei for the purposes of removing them from the other detectors of the LAT as background. Thus, from the perspective of the GEM, the primary purpose of the inputs provided by the ACD is as a trigger *veto*. However, to assist in absolute energy calibrations, a second set of signals is provided by the ACD called the CNO. These signals are used by the GEM as a *trigger*, as will be explained in the sections below.

As a detector, the ACD consists of a set of 97[1] scintillating *tiles*. The light output of any one tile is captured *twice* by two individual Photo-Multiplier-Tubes (PMTs) and fibres in order to both satisfy instrument redundancy requirements and increase detector efficiency. Both fibres are envisioned as being always active; however, LAT electronics, at various points in its chain, has the ability to mask any one fibre when or if 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 tile's *A*-side and the other half as the tile's *B*-side.

The ACD Front-End electronics system consists of 12 printed circuit boards (the FREE board). Each FREE board contains two types of ASICs: the GAFE (GLAST ACD Front-End) and the GARC (GLAST ACD Readout Controller). One fiber is serviced by one GAFE. A FREE board contains 18 GAFEs which in turn are managed and serviced by a single GARC. This results in 216 electronics channels with the capability to service 108 fibre pairs. As the detector has only 97 tiles, this results in 11 *unassigned* fibre pairs. These unassigned pairs will be discussed separately below.

Each GAFE splits the light output of its PMTs into two signals: a *veto* signal and a CNO signal. The CNO signal when processed on the FREE board is referred to as the HLD (High Level Discriminator) signal. Both the *veto* and HLD signals are separately amplified, shaped and discriminated. The threshold for each discriminator is configured through a configuration register of the GAFE. The digital output of each discriminator is maskable, again through a configuration register of the GAFE. The peak shaping time for each of the signals is identical and equal to 3.2 microseconds. This process is illustrated in Figure 5:

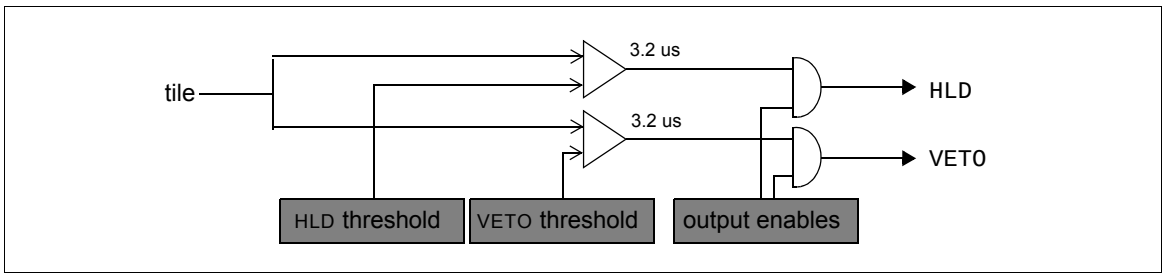

Figure 5 Forming the HLD and VETO outputs on the GAFE

1. If one discounts the ribbons, actually 89 tiles.

The CNO signals are "wire ORed" on each FREE board. Thus, there is one CNO signal per FREE board, for a total of twelve (12) CNO inputs delivered to the GEM. The GEM receives *veto* signals for each side of each tile. The signals from the *A* and *B* sides of any one tile are simply or'd together by the GEM to form *one* signal per tile. It is *only* this one signal which is both captured and recorded by the GEM.

Data is brought from each FREE board to the LAT on two cables, the information being duplicated on each cable. One cable from each FREE board goes to the *primary* GEM and the second cable goes to the *redundant* GEM. Consequently, it is convenient to think of any one GEM as receiving data on twelve cables; 6 for the *A* fibres and 6 for the *B* fibres.

Internally, the GEM simply numbers the FREE boards from zero (0) to eleven (11). The correspondence between this number and FREE board *name* is enumerated in Table 4:

Table 4 Relationship between FREE board number and FREE board name

Number	Name
0	1LA
1	1RB
2	2LA
3	2LB
4	2RA
5	2RB
6	3LA
7	3RB
8	4LA
9	4LB
10	4RA
11	4RB

The mapping between tile number, FREE board name and channel is detailed in Appendix A.

1.3.2 The Tower

The LAT contains *sixteen* towers. Each tower consists of two detectors: a silicon-strip based *Tracker* and a Cesium Iodide based *Calorimeter*. For each tower the tracker produces *one* Trigger Input to the GEM. This input is called the TKR. For each tower, the calorimeter produces *two* Trigger Inputs to the GEM. One is called the Calorimeter *Low* Energy (CAL_{LE}) Input and the other, the Calorimeter *High* Energy (CAL_{LH}) Input. These inputs are synthesized by a combination of tracker and calorimeter based electronics and the Tower Electronics Module (TEM). The TEM is described in detail in [6]; however, the following sections describe the

behaviour of both detectors and their associated electronics from the perspective of synthesizing the inputs used by the GEM in forming a trigger decision.

1.3.2.1 The Calorimeter

For a single tower, the Calorimeter consists of 96 *CsI* logs arranged as lincoln logs of 12 columns and 8 rows, with alternating *x* and *y* orientation, i. e., 4 *x*-layers and 4 *y*-layers of twelve logs apiece. Each log end is serviced separately by a custom ASIC called the GLAST Calorimeter Front End (GCFE). Thus, the calorimeter electronics uses a total of 192 GCFEs. One end of a layer is called its "positive" end and the other its "negative" end. The GCFE process signals from two diodes connected to the crystal, a small area diode to cover the high energy regime and a large area diode[1] to cover the low-energy regime. These signals are split out to provide support for both trigger and event; however, this document will concentrate only on the signal processing germane to the trigger. The signal from each diode is amplified, shaped, and discriminated. The threshold for each discriminator is configured through a register of the GCFE. The digital output of each discriminator is maskable, again through a configuration register of the GCFE. It is these two signals: one a low energy signal (`LE_DISC`) and one high energy (`HE_DISC`) which are output by each and every GCFE. The shaping time for both of these signals is 500 nanoseconds. This process is illustrated in Figure 6:

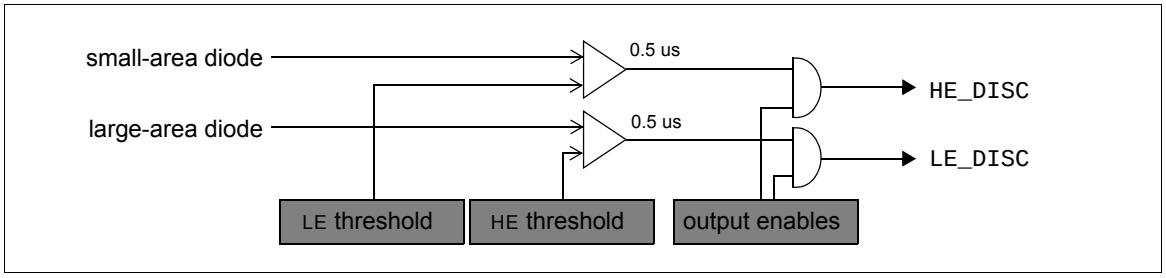

Figure 6 Forming the LE_DISC and HE_DISC outputs on the GCFE

A *layer*-end worth of logs (12) is serviced by another custom ASIC called the GLAST Calorimeter Readout Controller (GCRC). Each trigger output of each GCFE is "wire ORed" together to present two signals to the GCRC: one the High-energy sum for the layer-end and the other, the Low Energy sum for the layer-end. A GTRC receives these two sums, registers these signals with the system clock (`sysclk`) and forwards them for further processing to the TEM. Four GCRCs along with their 48 GCFEs are carried on a single circuit board called the Calorimeter Analog Front-End Electronics (AFEE) and in actuality, it is this board which interfaces to the TEM. Thus, the AFEE board sends *eight* Trigger Requests to the TEM: four High-Energy requests (`NTREQHEx`) and four Low-Energy requests (`NTREQLEx`). Each request corresponds to the sum for one layer-end. This process is illustrated in Figure 7:

1. 4 times the size of the small area diode.

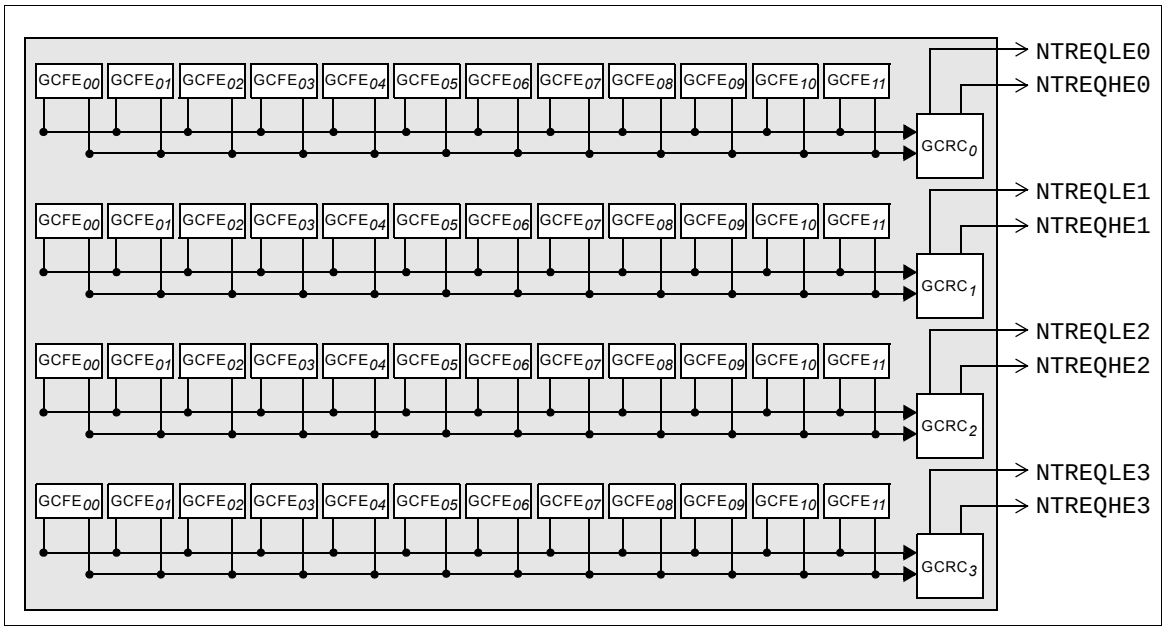

Figure 7 Forming the eight Calorimeter Trigger Requests on the AFEE board

As there are eight layers and each AFEE board services four layer-ends, a total of four boards must interface to the TEM in order to service the entire calorimeter. The TEM receives a total of thirty-two Trigger Requests in two sets: 16 representing the High-Energy signal, and 16 representing the Low-Energy signal. The TEM processes each of these sets *identically* and therefore, this section will discuss only the processing of the *Low*-Energy Trigger Requests.

As the Trigger Request signals arrive at the TEM, they may be delayed by a configuration register in order to take out any timing variance between tracker and calorimeter based signals. (See [6].) Once in the TEM, processing begins by combining layer-ends together, in order to form *layer* signals. Each of the incoming layer-end requests are separately maskable through configuration registers of the TEM's four GCCCs (GLAST Calorimeter Cable Controller). Each one of the resulting eight outgoing layer signals are also maskable: this time through a configuration register of the TEM's GTIC (GLAST Trigger Interface Controller). These eight masked signals are then summed together to form *one* Calorimeter Low-Energy (CAL_{LE}) Trigger Input. It is this signal which is converted to differential (LVDS) and forwarded to the GEM. The same, exact process is applied to the High-Energy Trigger Requests to form *one* Calorimeter High-Energy (CAL_{HE}) Trigger Input. Thus, each tower sends *two* calorimeter based inputs to the GEM. Each signal is the digital OR over the calorimeter's eight layers. This process is illustrated for the Low-Energy Trigger Requests by Figure 8:

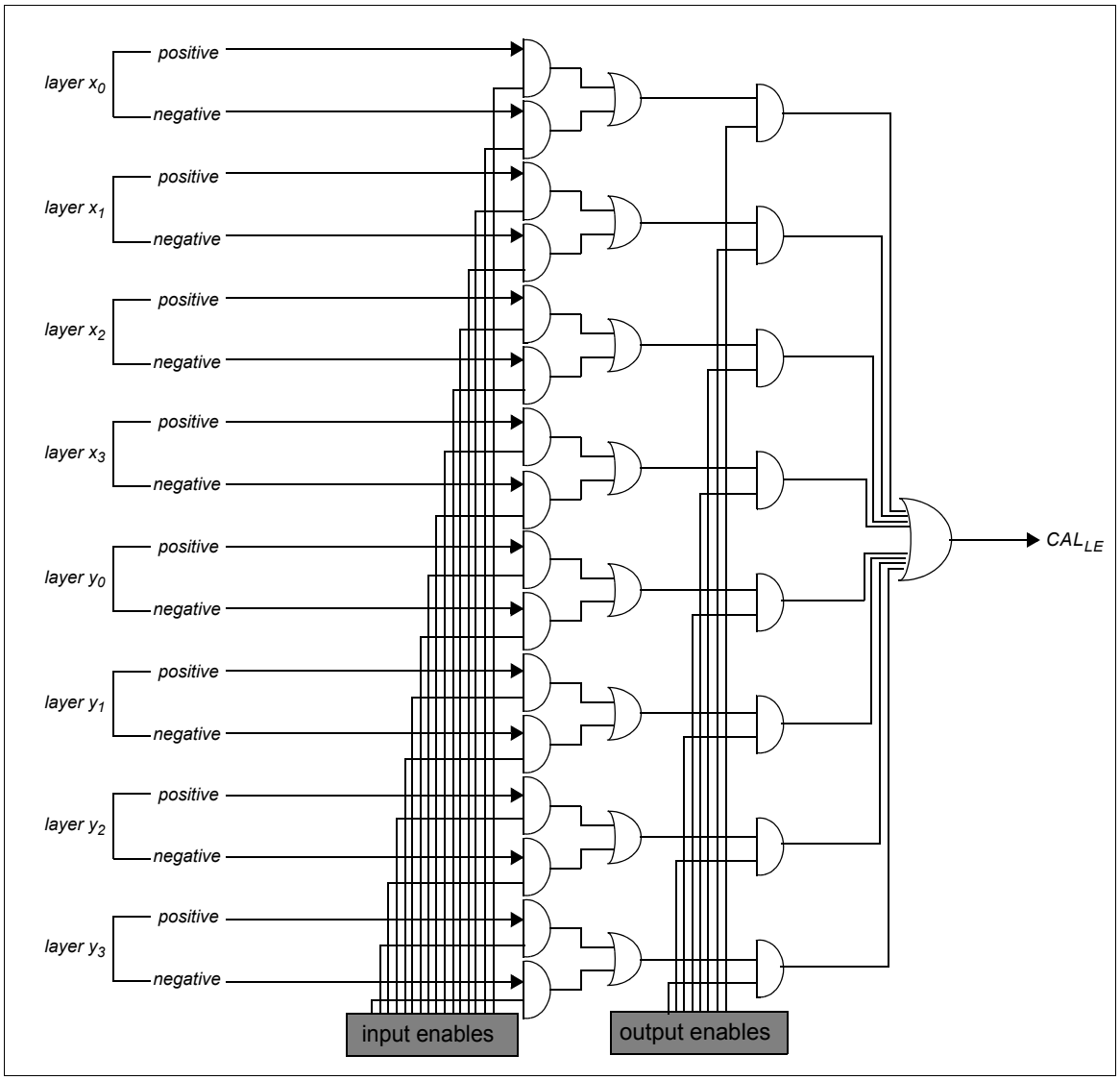

Figure 8 Forming of the two Calorimeter Trigger Inputs on the TEM

1.3.2.2 The Tracker

For a single tower, the Tracker consists of 36 silicon strip detectors arranged as 18 *x*-layers and 18 *y*-layers. Corresponding *x* and *y* layers will be called *layer pairs*. Each layer has 24 sections of 64 strips, or a total of 1536 strips/layer. Each section of a layer is serviced by a custom ASCI called the GTFE (GLAST Tracker Front-End). Each GTFE manages 64 strips. The 24 GTFEs of a layer are serviced by another custom ASIC called the GTRC (GLAST Tracker Readout Controller). For reasons of redundancy, a layer uses *two* GTRCs. Each GTRC nominally servicing half of the GTFEs of the layer; however, the split is arbitrary and one GTRC could, for example, service all 24 GTFEs of its layer. The combination of 24 GTFEs and 2 GTRCs are contained within a single (passive) structure called an MCM (Multi-Carrier-Module).

Information managed by the GTRC on one end of the MCM is called the *left layer-end* and information managed by the other GTRC, the *right* layer-end.

The GTFE receives analog information from 64 strips, which are then amplified, shaped, and discriminated, producing 64 digital signals. The threshold for discrimination is a configuration parameter of the GTFE. Note, however, there is only one threshold for all 64 channels, *not* one threshold per channel. A GTFE produces a single trigger related output using these digital signals called its "FAST-OR".[1] To produce this signal each of the 64 digital signals is masked, again through a configuration register of the GTFE. The 64 masked outputs are then ORed together. The output of this operation is then ORed against the FAST-OR of a *previous* GTFE to produce the *next* FAST-OR.

Note: The act of feeding back the previous FAST-OR causes a 5 Nano-second delay[2] in the development of the next FAST-OR.

This entire process is illustrated in Figure 9:

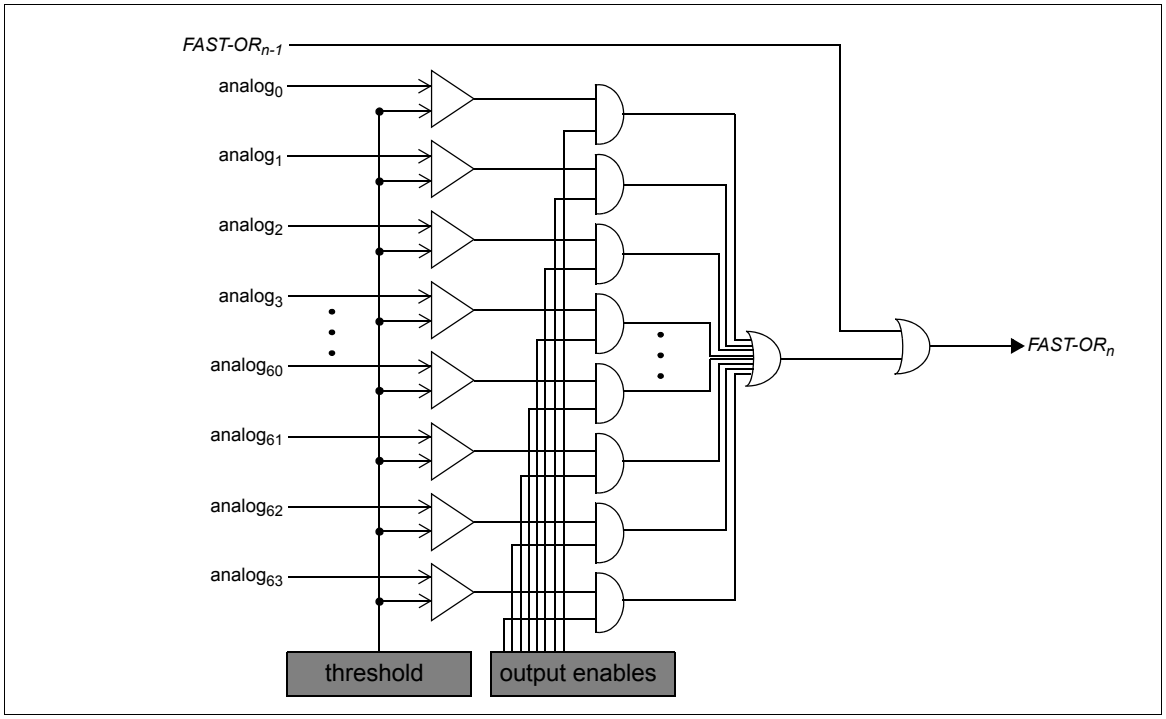

Figure 9 Forming the FAST-OR output on the GTFE

In this fashion all the GTFEs on a MCM are daisy chained together to form a single FAST-OR signal per layer-end. This FAST-OR signal is input to the appropriate GTRC for the layer-end, where, conditionally, some processing of the FAST-OR is done and the resulting signal sent to the TEM as a Tracker *Trigger Request*. This process is illustrated in Figure 10:

1. Note: The 64 digital signals are used as input for *both* event and trigger, however this document will focus only on their use by the trigger.
2. Older versions have a longer delay (as much as double). Need to verify which older versions.

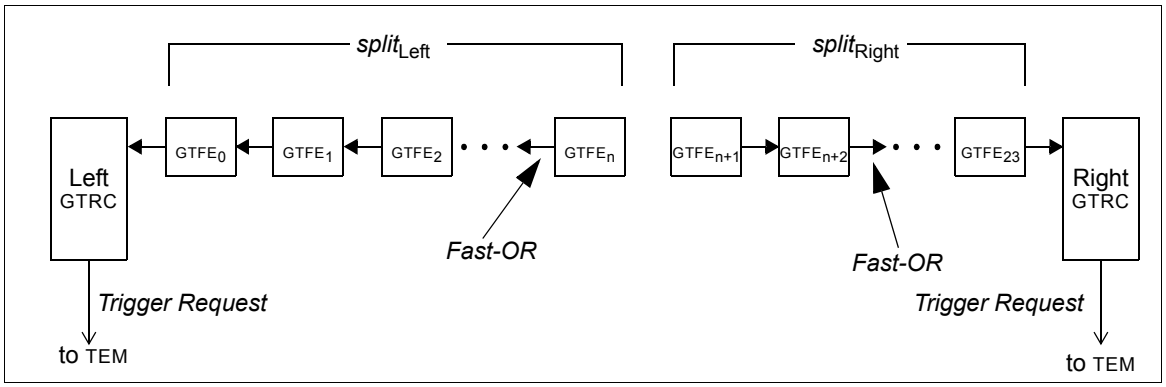

Figure 10 Generation of Trigger Requests on an MCM

Thus, the tracker contributes *seventy-two* Trigger Request signals to the TEM, one for each layer-end. This arrangement of synthesizing the Trigger Request lines has two consequences:

i. Because the Trigger Request signal goes through the GTRC, the GTRC will conditionally affect both the nature and shape of the Trigger Request signal. Its effect is determined by the value of the `STRETCH_OR` field of the control register. (See [7].)
   - If the value is *zero* (0), the signal is simply passed through with only a 1/2 to 3/2 `sysclk` delay.
   - If the value is *non*-zero, the signal is both de-glitched and stretched. The value of this field determines the amount of stretch (in units of `sysclk`).

ii. Daisy-chaining has some unfortunate side-effects:
   - Increased Trigger Request latency
   - Additional Trigger Request jitter

Note: The magnitude of these side-effects depends on where in the MCM a signal is developed and consequently on how the layer is split between left and right.

As the Trigger Request signals arrive at the TEM, they may be delayed by a configuration register in order to take out any timing variance between tracker and calorimeter based signals. (See [6].) Once in the TEM, the 72 left and right end Trigger Request Signals are `OR`ed into 36 *layer* signals. It is these signals which form the input to the TEM's "3-in-a-row" algorithm. This algorithm looks for a three-fold coincidence between any 3 adjacent layers of a tower. To forestall a dead layer from impacting the algorithm's efficiency, the TEM provides a mechanism to allow a user to substitute for the absence of a layer. This is accomplished by `OR`ing the contents of a configuration register with the 36 layer signals *before* input to the "3-in-a-row" logic. Given these inputs, the algorithm can locate sixteen possible "3-in-a-row" combinations. These combinations are enumerated in Table 5. The output of each combination can be enabled or disabled through a configuration register of the TEM. Once formed and suitably masked, the 16 combinations are summed. This signal is then converted to differential (LVDS) and sent to the GEM as the TKR Trigger Input for a particular tower. This process is illustrated in Figure 11:

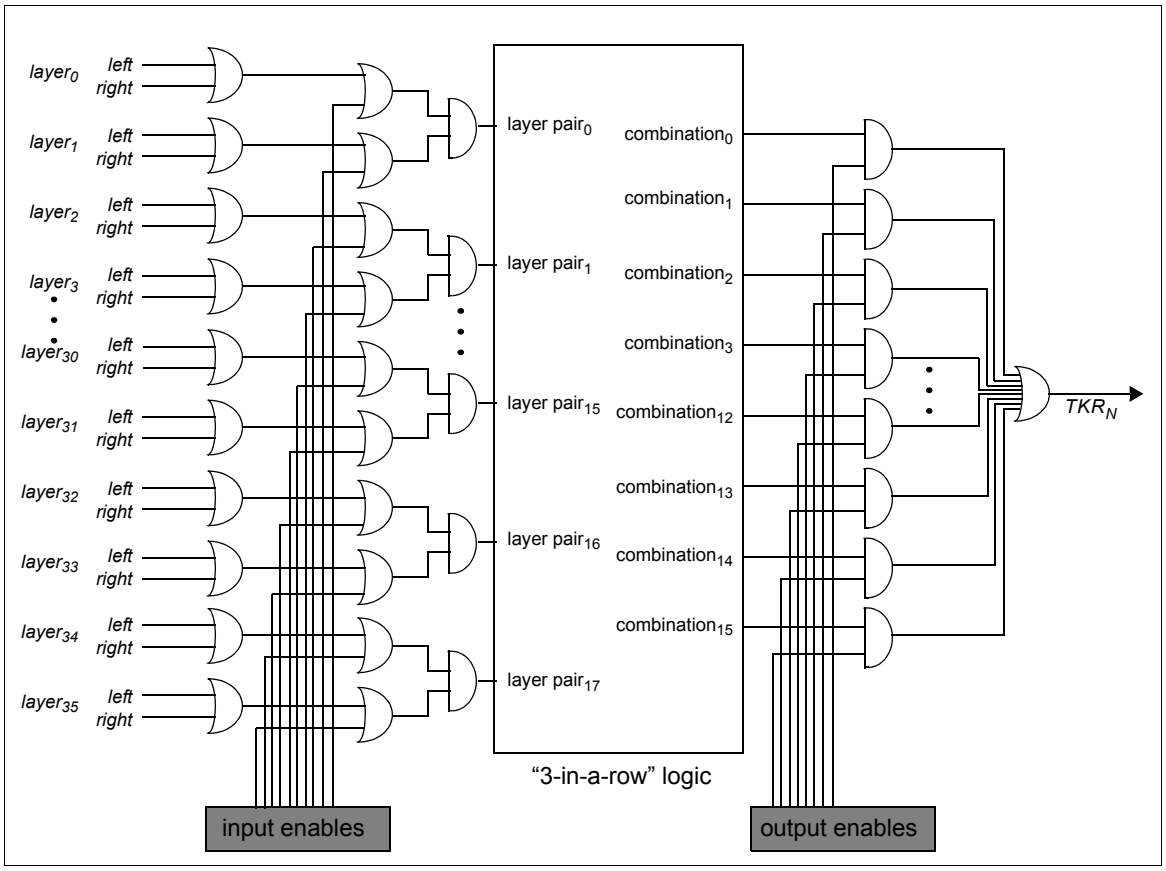

Figure 11 Forming the TKR Trigger Input on the TEM

And it is this result which is forwarded by the TEM to the GLT for further processing. These management registers are described in the following sections.

Table 5 Possible combinations of TEM "3-in-a-row" Trigger Requests

Combination	Coincidence of layer pairs...		
0	$[y_0, x_0]$	$[y_1, x_1]$	$[y_2, x_2]$
1	$[y_1, x_1]$	$[y_2, x_2]$	$[y_3, x_3]$
2	$[y_2, x_2]$	$[y_3, x_3]$	$[y_4, x_4]$
3	$[y_3, x_3]$	$[y_4, x_4]$	$[y_5, x_5]$
4	$[y_4, x_4]$	$[y_5, x_5]$	$[y_6, x_6]$
5	$[y_5, x_5]$	$[y_6, x_6]$	$[y_7, x_7]$
6	$[y_6, x_6]$	$[y_7, x_7]$	$[y_8, x_8]$
7	$[y_7, x_7]$	$[y_8, x_8]$	$[y_9, x_9]$
8	$[y_8, x_8]$	$[y_9, x_9]$	$[y_{10}, x_{10}]$
16			

Table 5 Possible combinations of TEM "3-in-a-row" Trigger Requests

Combination	Coincidence of layer pairs...		
9	$[y_9, x_9]$	$[y_{10}, x_{10}]$	$[y_{11}, x_{11}]$
10	$[y_{10}, x_{10}]$	$[y_{11}, x_{11}]$	$[y_{12}, x_{12}]$
11	$[y_{11}, x_{11}]$	$[y_{12}, x_{12}]$	$[y_{13}, x_{13}]$
12	$[y_{12}, x_{12}]$	$[y_{13}, x_{13}]$	$[y_{14}, x_{14}]$
13	$[y_{13}, x_{13}]$	$[y_{14}, x_{14}]$	$[y_{15}, x_{15}]$
14	$[y_{14}, x_{14}]$	$[y_{15}, x_{15}]$	$[y_{16}, x_{16}]$
15	$[y_{15}, x_{15}]$	$[y_{16}, x_{16}]$	$[y_{17}, x_{17}]$
16			

1.3.3 Internal

There are three non-detector based sources which contribute Trigger Inputs to the GEM:

The Command/Response system: This system is the principal mechanism by which a user of the LAT interacts with the registers and functionality of the LAT's electronic modules. (See [3].) One of these functions on the GEM is the ability to force, or *solicit* a trigger. This function is described in more detail in Section 3.3.1.

The system clock: The common system clock (`sysclk`) is used by the GEM, potentially in conjunction with the 1-PPS signal to synthesise the periodic Trigger Input.The `sysclk` is a nominally 20 MHZ signal with a equally balanced duty cycle. In order to synthesize this signal, `sysclk` actually originates *off* the GASU, within one of the two SIU crates as a 40 MHZ signal. The 40 MHZ clock is input to the GASU, where it is divided down by two by the CRU (as discussed in [3]) and then redistributed to all the electronics of the LAT.

The spacecraft: The LAT is delivered a message from the spacecraft describing an absolute time based on its GPS system. This message is conventionally stated as "at the *tone* the time will be," where the tone corresponds to a signal originating from the spacecraft called the 1-PPS (One, Pulse-Per-Second) signal. The GPS message predates its corresponding 1-PPS signal by approximately 800 milliseconds. The 1-PPS signal is, as its name would imply, a precise and accurate (within +/- 1.5 microseconds, as discussed in [8]) time hack sent once, by the spacecraft, every second. This signal serves two purposes within the GEM:

— Strobes the internal time-base of the GEM, thus allowing an accurate measure of the time at which any event is produced relative to the 1-PPS signal.

— Is used (conditionally) in conjunction with the system clock to produce the Periodic Trigger Input. This function is intended as a diagnostic in order to actually test the precision and accuracy of both the 1-PPS signal and the LAT's system clock.

1.4 Trigger Inputs

All Trigger Inputs are received as LVDS signals. The first phase in trigger formation requires:

- Conversion of all inputs from differential to single-ended.
- Masking of all inputs against a user specified configuration.
- Consolidation to remove signal redundancy. The consolidation includes:
  - ORing the *tile* signals of the *A* and *B* tile sides from the ACD
  - ORing the *busy* signals from the sixteen towers and AEM.

When this phase is complete, the GEM is left with the following "raw" inputs:

- 108 *tile* signals from the ACD (eighteen per FREE board)
- 12 CNO signals from the ACD (one per FREE board)
- 16 CAL_{LE} signals from the calorimeter (one per tower)
- 16 CAL_{HE} signals from the calorimeter (one per tower)
- 16 TKR signals from the tracker (one per tower)
- 1 *busy* signal representing the busy state of the entire LAT

This phase is accomplished using a custom ASIC called the GLTC. The GEM requires 18 of these ASICs to process the entire set of input signals. Each ASIC has 18 identical "channels", each channel providing a maskable, differential (LVDS) to single-ended (CMOS) conversion for a single input. In addition to providing conversion, the ASIC also ORs adjacent channels as illustrated in Figure 12:

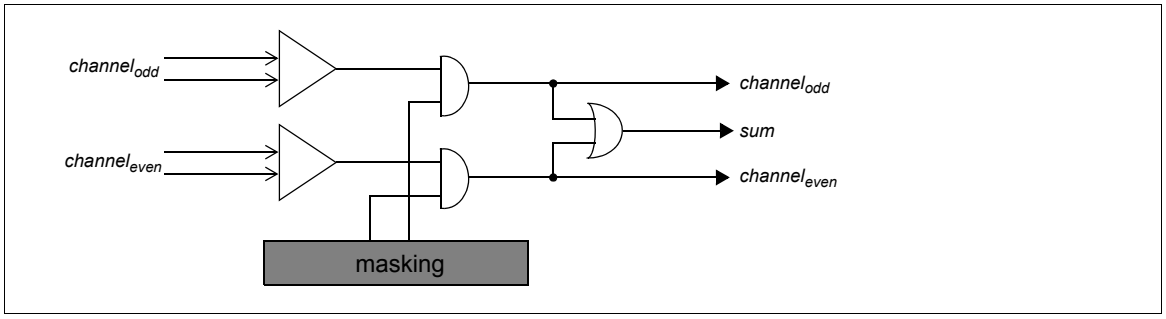

Figure 12 One channel of the GLTC ASIC

Finally, the ASIC provides one output representing the sum over *all* the channels. The masking for each chip is brought out through the registers described in Section 2.9.

In addition to the "raw" inputs, the GEM has three additional supplementary inputs:

- The system clock (`sysclk`), nominally operating at 20 MHZ
- The once-per-second signal (1-PPS) sent by the spacecraft
- The *solicited* trigger, initiated by a user through the Command/Response interface

These signals are described in more detail in Section 1.5. The 108 *tile* inputs are brought to the ROI generator for further processing to create the 16 ROI Input signals. This processing is discussed in the following section.

1.4.1 Tiles, Vetos and Region of Interests

The principal function of the ROI generator is to take the 108[1] tile signals and reduce these inputs to 16 Region-Of-Interest (ROI) signals. These sixteen signals are used by the Scheduler for two purposes:

- To open a trigger window. (See Section 1.5.1.)
- To form the ROI condition. This condition depends on how the GEM is configured to interpret the results from its ROI generator, either as a:
  - — Set of tower shadows in order to *veto* a trigger decision. (See Section 1.7.1.)
  - — Set of coincidences in order to *form* a trigger decision. (See Section 1.7.1.)

A Region-Of-Interest is simply a user defined subset of the entire 108 tiles of the ACD. The GEM allows the user to define as many as *sixteen* ROIs. Any one ROI may include as few as *none* (the null set) or as many as *all* (the entire set of 108 tiles). The subsets may overlap; that is, a tile may be included in more then one subset. The user defines for each tile in which subsets they are included through a series of configuration registers of the ROI generator as described in Section 2.8. The ROI *signal* is simply whether any *one* of the tile signals defined in the ROI is asserted. The process of defining an ROI and forming its corresponding ROI signal is illustrated in Figure 13:

1. Where one includes both the ribbons and NA channels as tiles.

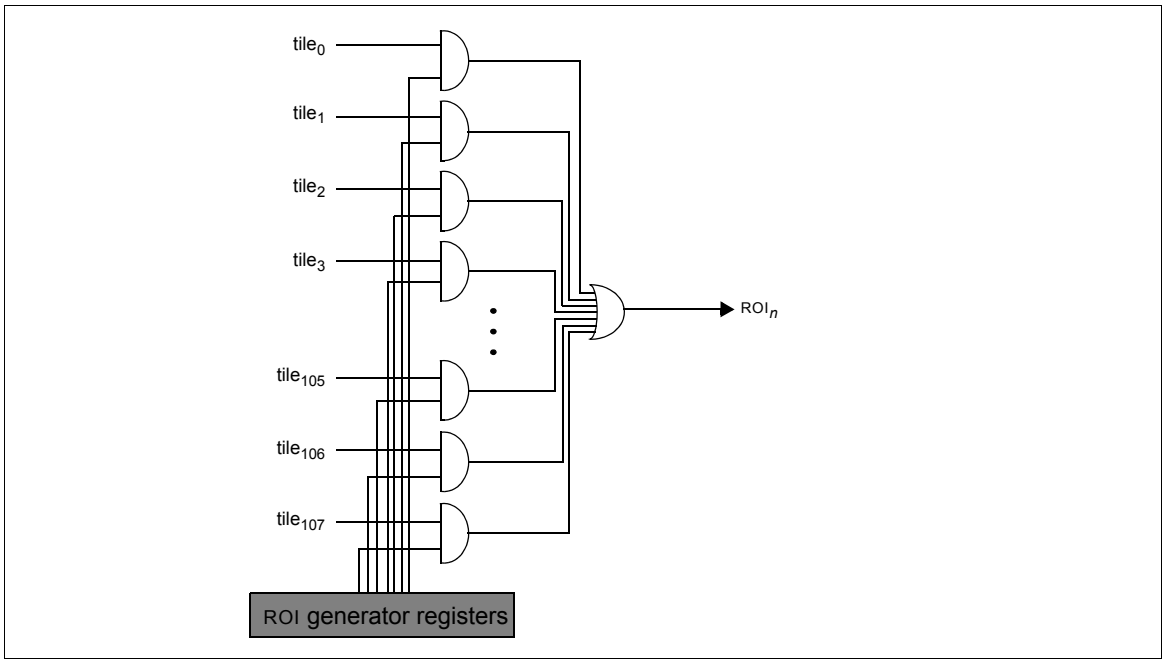

Figure 13 Forming one of sixteen ROIs (Region of Interest)

Each one of these sixteen engines receives the same input (the 108 tiles) and each engine produces one output. Thus, the input to the ROI generator of 108 tiles produces an output of 16 ROI signals as illustrated in Figure 14. These signals are then used as input by the Scheduler as described in more detail in the following sections.

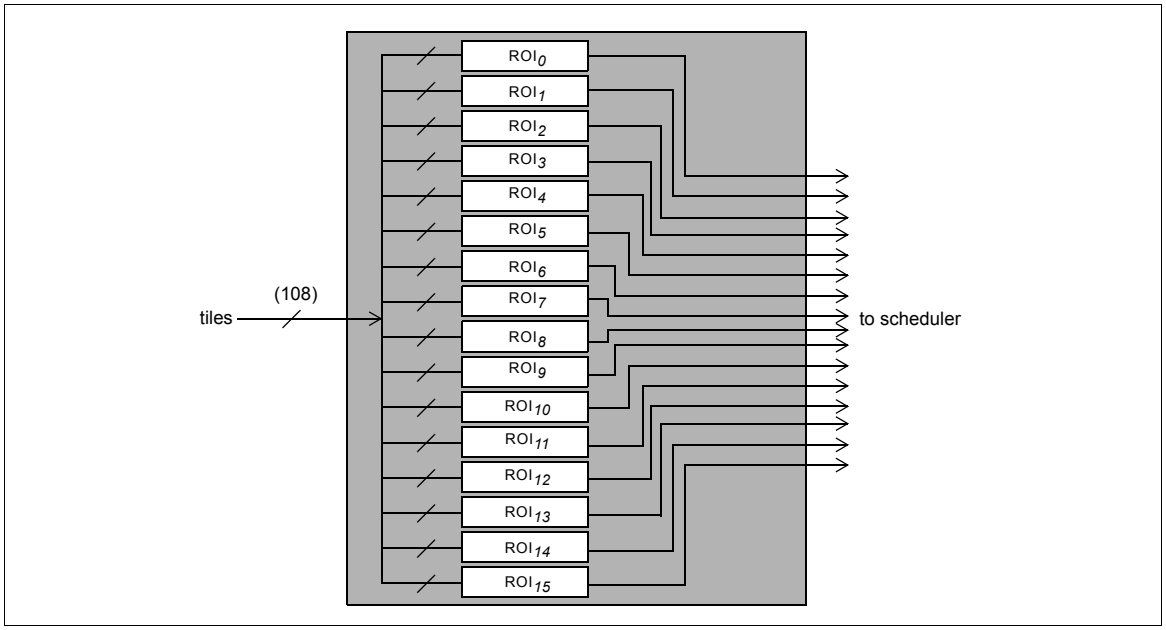

Figure 14 Forming the sixteen ROI signals

1.5 The Trigger Window

Need a good explanation of the trigger window and its purpose. To be written.

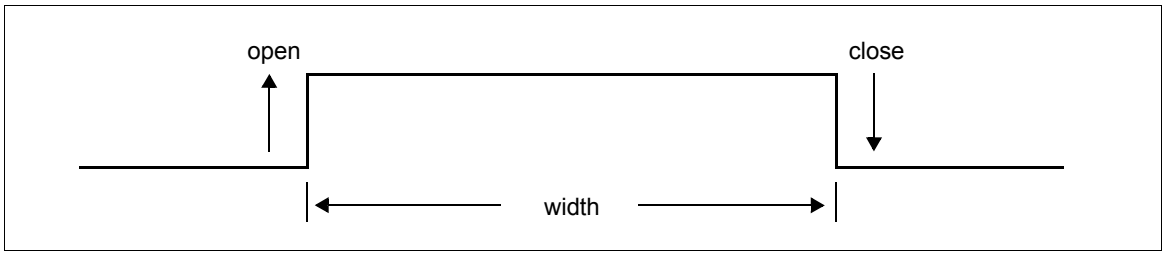

Figure 15 The window signal

A trigger window is opened by a *transition* (from logically false to logically true) of any one of seven conditions *or* the ROI signal. The ROI signal is defined as one or more of the sixteen ROIs asserted by the ROI generator. (See Section 1.4.1.) The seven conditions are defined as:

Tracker: One or more of the TKR signals asserted by the sixteen towers.

Calorimeter (Low Energy): One or more of the CAL_{LE} signals asserted by the sixteen towers.

Calorimeter (High Energy): One or more of the CAL_{HE} signals asserted by the sixteen towers.

CNO: One or more of the CNO signals asserted by the 12 FREE boards of the ACD.

periodic: A periodic, fixed rate signal. The frequency of this signal is derived from either the system clock or the 1-PPS signal. (See Section 2.3.3.)

solicited: The receipt by the GEM of the dataless `TRIGGER` command. (See Section 3.3.1.)

external: A transition is asserted on the external trigger signal carried into the GASU through its test connector (see [5]).

The synthesis of each of these seven conditions and one ROI signal is described in more detail in the sections below. Each one of these conditions and the ROI signal can be individually masked through a configuration register (discussed in Section 2.3.1), *before* the logic applied to open a window. Writing a value of *zero* (0) to this register is the simplest and most straight-forward way to put the GEM in a quiescent state such that its other registers may be modified without introducing side-effects.

The amount of time the window remains open is called the window *width*. The window width is independent of which of the six conditions or ROI signal were responsible for its opening, instead, the width is configurable with its value determined by a configuration register. (See Section 2.4.1.). The minimum width of the window is *one* (1) system clock. The maximum width of the window is *thirty-one* clock tics (31) system clocks.

Due to the finite recovery time of the GEM, the minimum time between any two adjacent windows is *two* system clocks (nominally 100 nanoseconds). This recovery time is called the "dead zone". If either a condition or the ROI signal is asserted during this interval it will *not* result in a window turn. Consequently, the number of times this occurs amounts to a dead-time correction. In order to facilitate this correction, the GEM counts the number of times a condition is asserted in the dead-zone (see Section 2.6.1.2).

The minimum width of the window is *one* (1) system clock. The process of forming a trigger window is illustrated in Section 16:

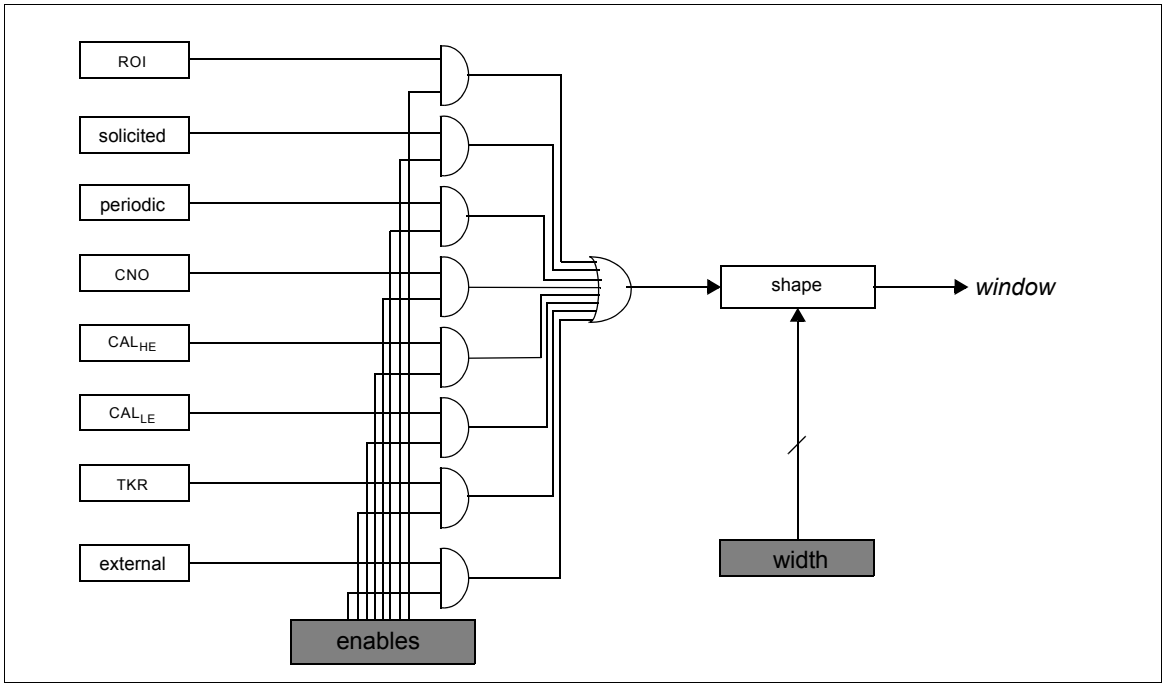

Figure 16 Forming the window signal

1.5.1 The ROI Signal

The ROI generator generates sixteen Region-Of-Interest Trigger Inputs. (See Section 1.4.1.) This signal is simply the assertion of one or more of these inputs. The synthesis of this signal is represented by the schematic of Figure 17:

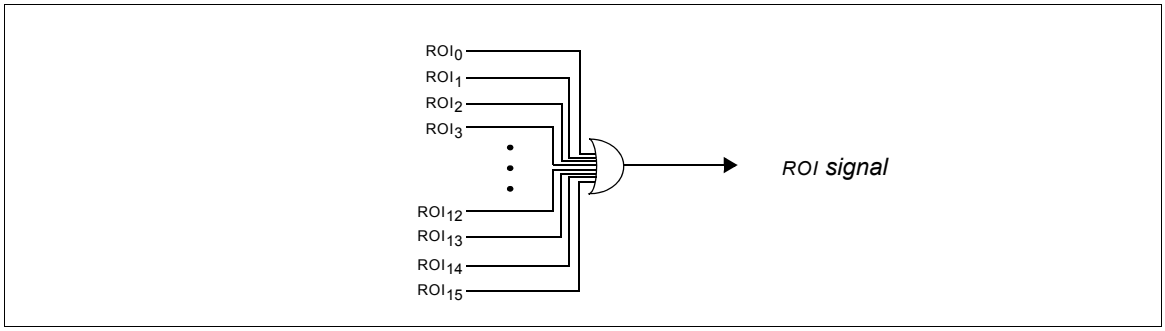

Figure 17 Forming the Region-Of-Interest (ROI) Signal

1.5.2 The Tracker Condition

Each tower (through its TEM) supplies a single TKR Trigger Input. This condition is simply the assertion of one or more of these inputs from the sixteen towers of the LAT. The synthesis of this condition is represented by the schematic of Figure 18:

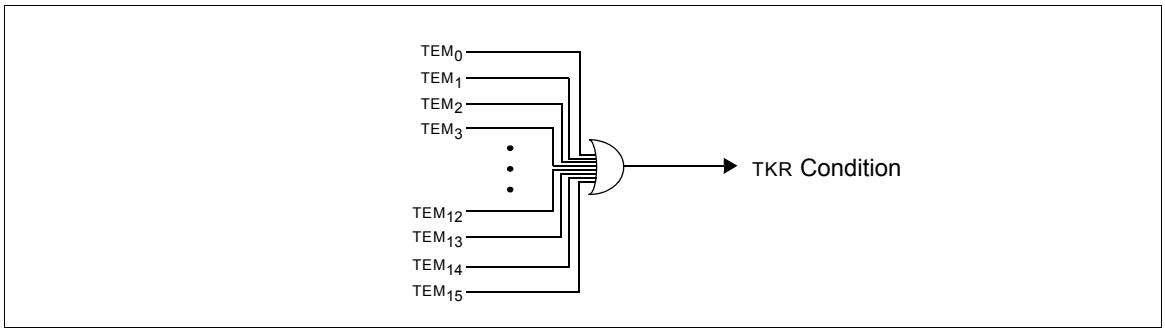

Figure 18 Forming the Tracker (TKR) Condition

1.5.3 The Calorimeter (Low Energy) Condition

Each tower (through its TEM) supplies a single Calorimeter *Low* Energy (CAL_{LE}) Trigger Input. This condition is simply the assertion of one or more of these inputs from the sixteen towers of the LAT. The synthesis of this condition is represented by the schematic of Figure 19:

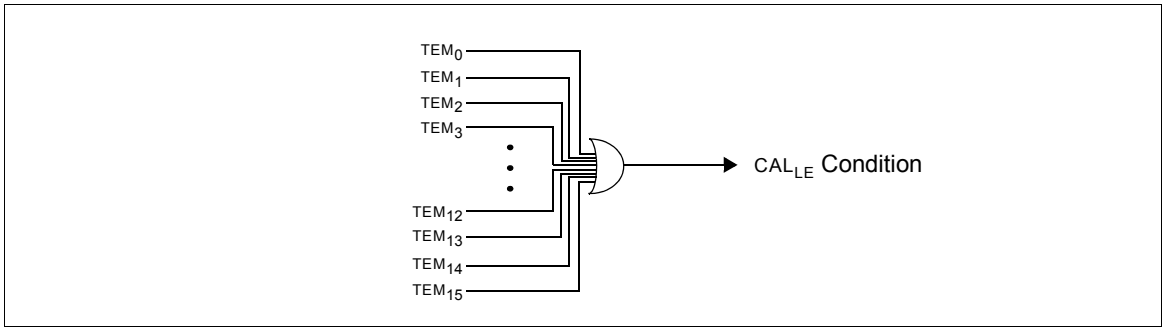

Figure 19 Forming the Calorimeter (Low Energy) Condition

1.5.4 The Calorimeter (High Energy) Condition

Each tower (through its TEM) supplies a single Calorimeter *High* Energy (CAL_{HE}) Trigger Input. This condition is simply the assertion of one or more of these inputs from the sixteen towers of the LAT. The synthesis of this condition is represented by the schematic of Figure 20:

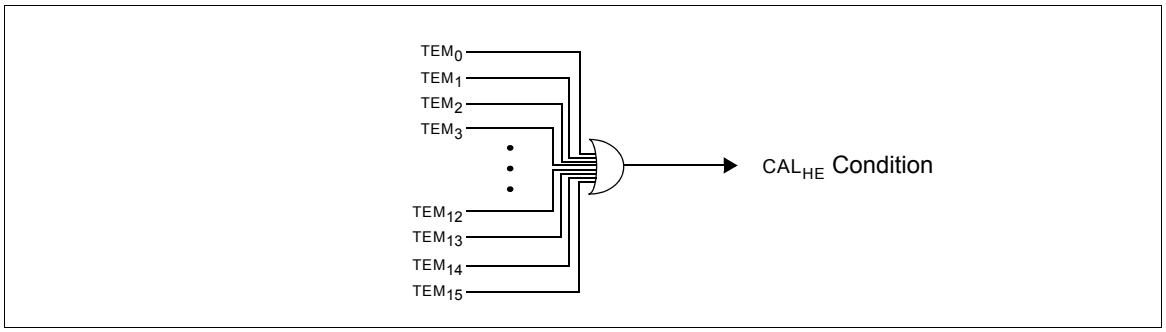

Figure 20 Forming the Calorimeter (High Energy) Condition

1.5.5 The CNO Condition

Each FREE board of the ACD supplies a single Trigger Input. This condition is simply the assertion of one or more of these inputs from the twelve boards comprising the ACD. The synthesis of this condition is represented by the schematic of Figure 21:

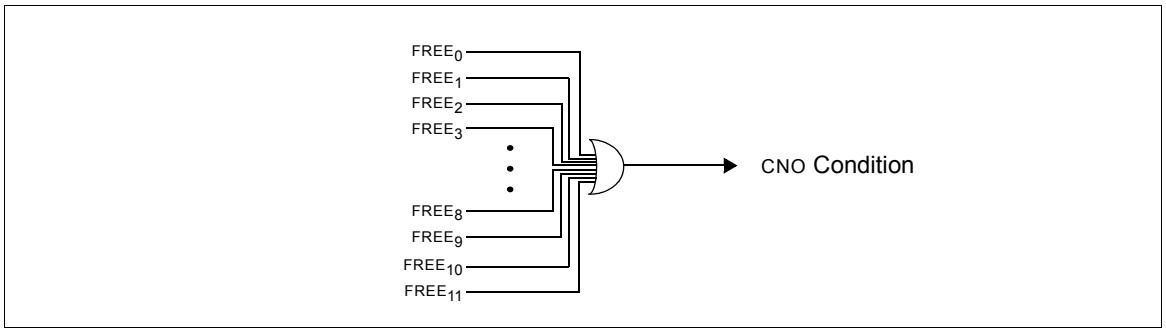

Figure 21 Forming the CNO Open Window Signal

1.5.6 The Periodic Condition

The periodic condition is designed to meet two needs:

- During data-taking to obtain unbiased event samples at continuous, fixed intervals
- During calibration to inject charge at fixed intervals a finite number of times

Although both needs require a condition occurring at periodic intervals, the rate at which the condition occurs differs between the two requirements; consequently, the periodic condition rate must be programmable. There are two sources for the periodic condition: the system clock (`sysclk`) and the 1-PPS signal. The GEM allows the user to select one of these two sources through a configuration register. (See Section 2.3.3.1.) Typically, `sysclk` is used as the source, but using the 1-PPS signal as a source plays the two timing sources against one another, allowing testing of their respective stability and accuracy. After selecting a source it is then selectively slowed down using a down counter. The value for the countdown is

determined by a configuration register. (See Section 2.3.3.1.) Once the periodic condition has been developed the user has options:

- Limit, or bound, the number of periodic conditions delivered
- Allow a continuous, uninterrupted stream of periodic conditions

Which particular option (bound or continuous) is determined by a configuration register. (See Section 2.3.3.2.) If the periodic trigger condition is bound, the number of triggers is determined by yet another configuration register. (See Section 2.3.3.3.) The entire process of developing the periodic trigger condition is illustrated in Figure 22:

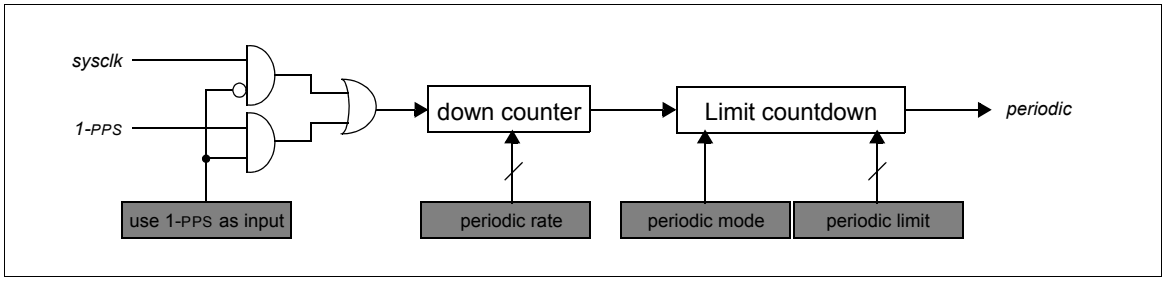

Figure 22 Forming the Periodic Open Window Signal

1.5.7 The Solicited Condition

This condition is asserted on the receipt of a dataless `TRIGGER` command received by the Command/Response Interface. (See Section 3.3.1.) Each instance of each command will generate one and only one Solicited condition.

1.5.8 The External Condition

This condition is asserted whenever the external trigger signal is asserted through the test connector of the GASU. See "gasu Based Teststands - A hardware and software Primer," Michael Huffer, lat-td-03664. for more information.

1.6 Forming the Trigger Vectors and Window Summary

While a window is open, the GEM's Scheduler latches the state of both its Trigger Inputs and the six conditions and ROI signal. This latched information is organized into five different *Trigger Vectors* and one *Window Summary*. The five vectors are defined as follows:

ROI vector: The sixteen Regions-Of-Interest from the ROI generator.

TKR vector: The TKR input from each of the sixteen towers.

CAL(Low Energy) vector: The CAL_{LE} input from each of the sixteen towers.

CAL(High Energy) vector: The CAL_{HE} input from each of the sixteen towers.

CNO vector: The CNO input from each of the twelve FREE boards.

These vectors are described in detail in the sections below. The Window Summary is a transient structure which summarizes the state of the seven conditions and ROI signal while the window was open. Both the Window Summary and TKR vector are used to synthesize the *Condition Summary* as described in Section 1.7. Abstractly, the function of synthesizing both vectors and summary can be represented by the FSM (Finite State Machine) illustrated in Figure 23. At any point in time, the engine is either in an *idle* or *latching* state. A window opening causes the engine to latch the Trigger Inputs and window signals into the five vectors and one summary. The presence of any one of these signals in a vector or summary specifies that the signal was present for at least one clock during the time the window remained open.

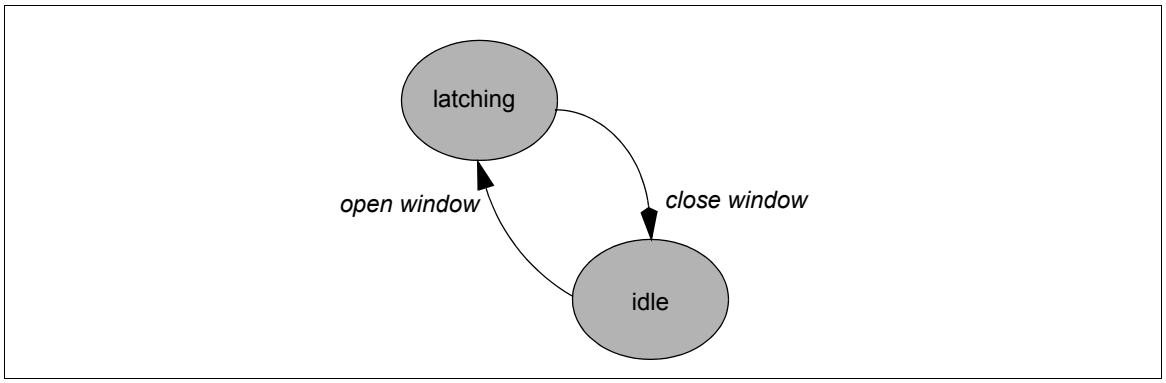

Figure 23 The Generic Latching Engine

1.6.1 The TKR Vector

The TKR Vector consists of the latched state of the sixteen TKR signals summed (one per tower) over the trigger window. The structure of this vector is illustrated in Figure 24. Each bit offset corresponds to a particular tower. If a bit at a particular offset is *set*, the corresponding tower had its TKR signal asserted for at least *one* system clock (20 MHZ) within the trigger window.

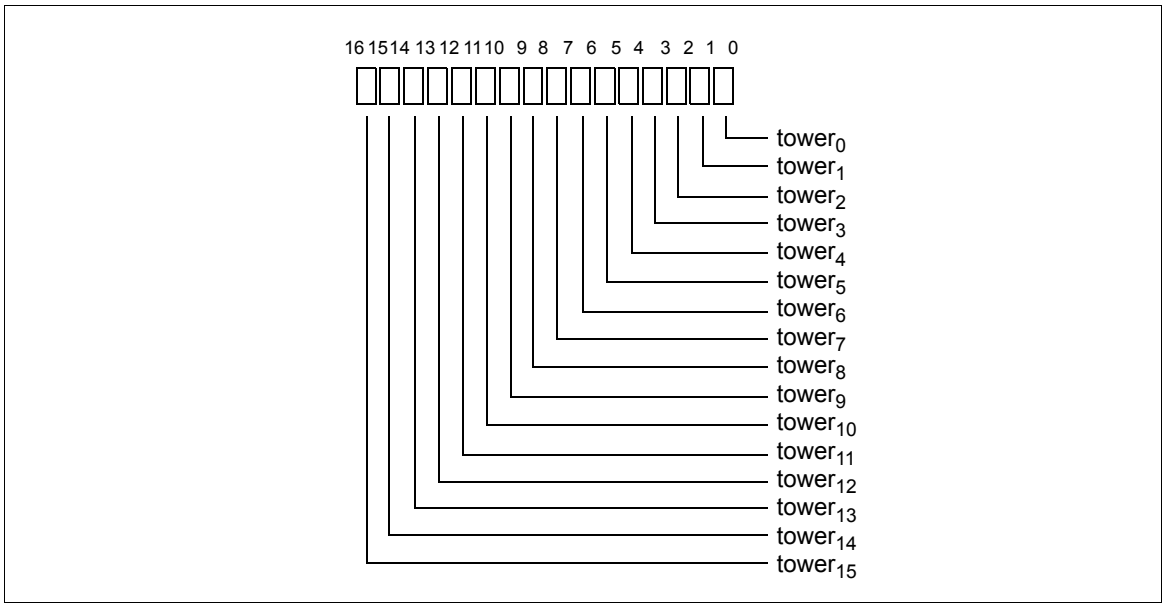

Figure 24 TKR Vector

1.6.2 The ROI Vector

The ROI Vector consists of the latched state of the 16 region signals produced by the ROI generator and summed over the trigger window. The structure of this vector is illustrated in Figure 25. If a bit at a particular offset is *set*, the corresponding region was asserted for at least *one* system clock (20 MHZ) within the trigger window. The interpretation of a region depends on whether the GEM was using regions as a *veto* or as a *trigger*. (See Section 1.7.)

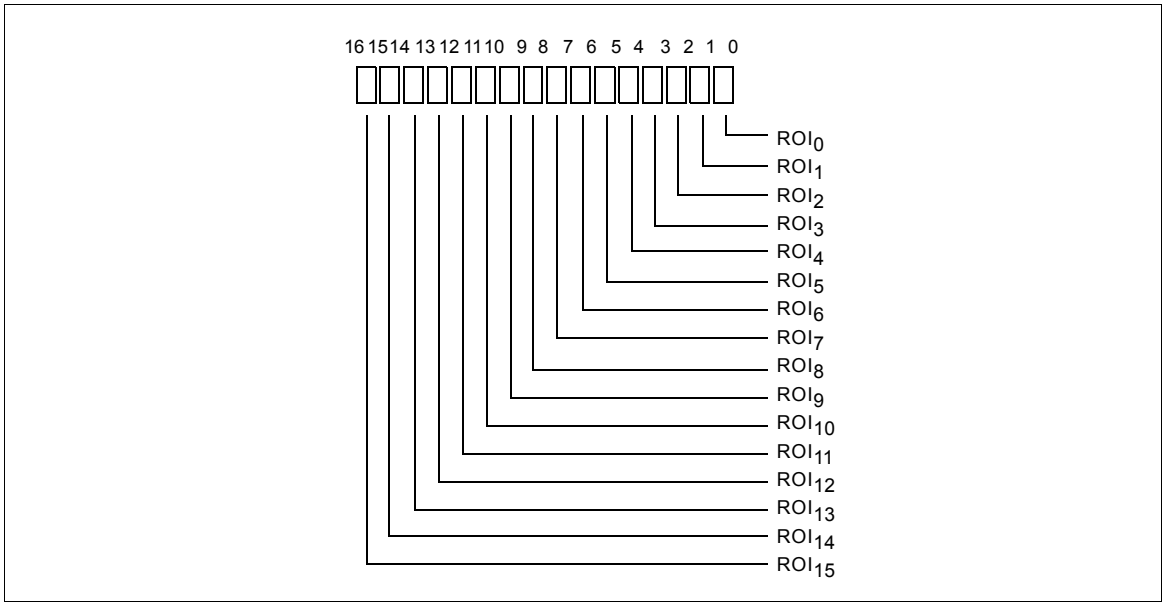

Figure 25 ROI Vector

1.6.3 The Calorimeter (Low Energy) Vector

The $\mathsf{CAL}_{\mathrm{LE}}$ Vector consists of the latched state of the sixteen $\mathsf{CAL}_{\mathrm{LE}}$ signals (one per tower) summed over the trigger window. The structure of this vector is illustrated in Figure 26. Each bit offset corresponds to a particular tower. If a bit at a particular offset is *set*, the corresponding tower had its $\mathsf{CAL}_{\mathrm{LE}}$ signal asserted for at least *one* system clock (20 MHZ) within the trigger window.

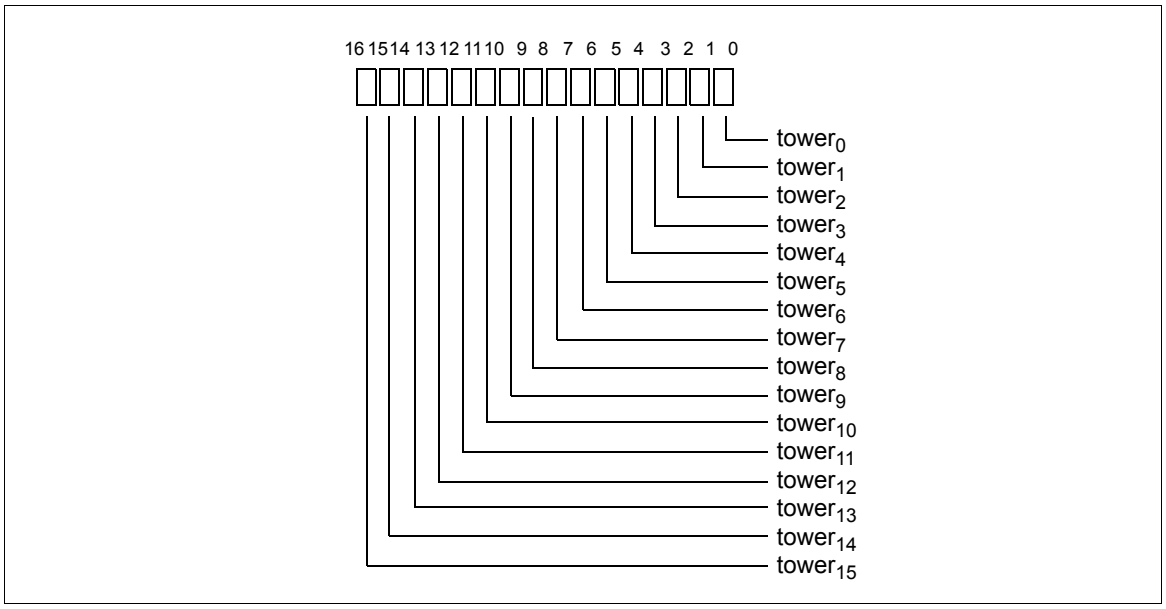

Figure 26 Calorimeter (Low Energy) Vector

1.6.4 The Calorimeter (High Energy) Vector

The $\mathsf{CAL}_{\mathrm{HE}}$ Vector consists of the latched state of the sixteen $\mathsf{CAL}_{\mathrm{HE}}$ signals (one per tower) summed over the trigger window. The structure of this vector is illustrated in Figure 27. Each bit offset corresponds to a particular tower. If a bit at a particular offset is *set*, the corresponding tower had its $\mathsf{CAL}_{\mathrm{HE}}$ signal asserted for at least *one* system clock (20 MHZ) within the trigger window.

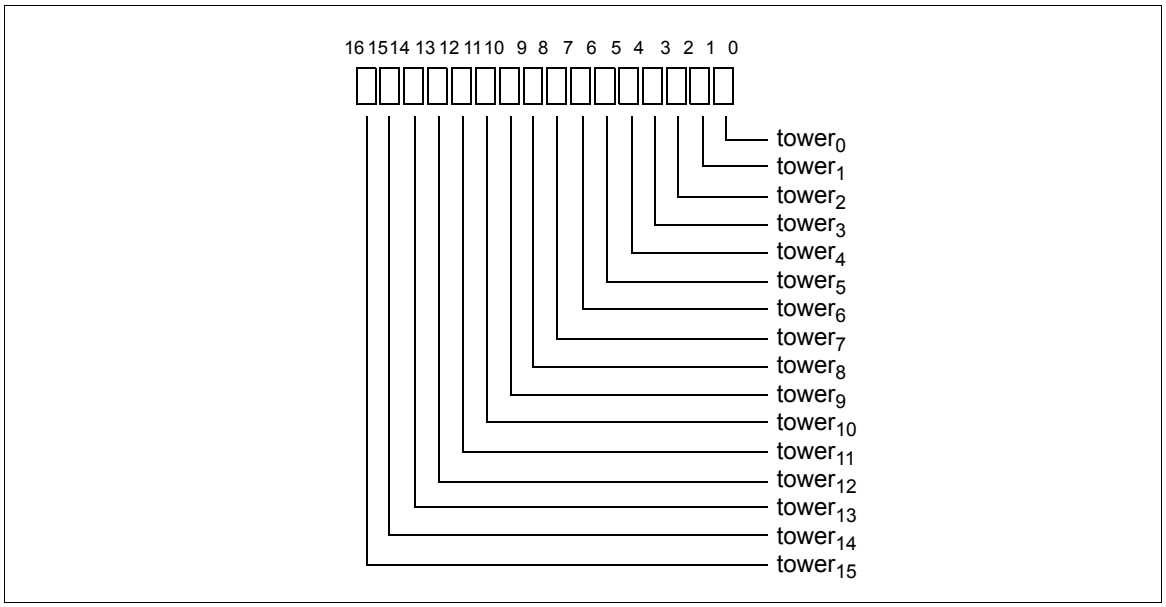

Figure 27 Calorimeter (High Energy) Vector

1.6.5 The CNO Vector

The CNO Vector consists of the sampled state of the twelve CNO signals produced by the ACD (one per FREE board) summed over the trigger window. The structure of this vector is illustrated in Figure 28. Each bit offset corresponds to a particular FREE board as enumerated in Table 4. If a bit at a particular offset is *set*, the corresponding FREE board had its CNO signal asserted for at least *one* system clock (20 MHZ) within the trigger window.

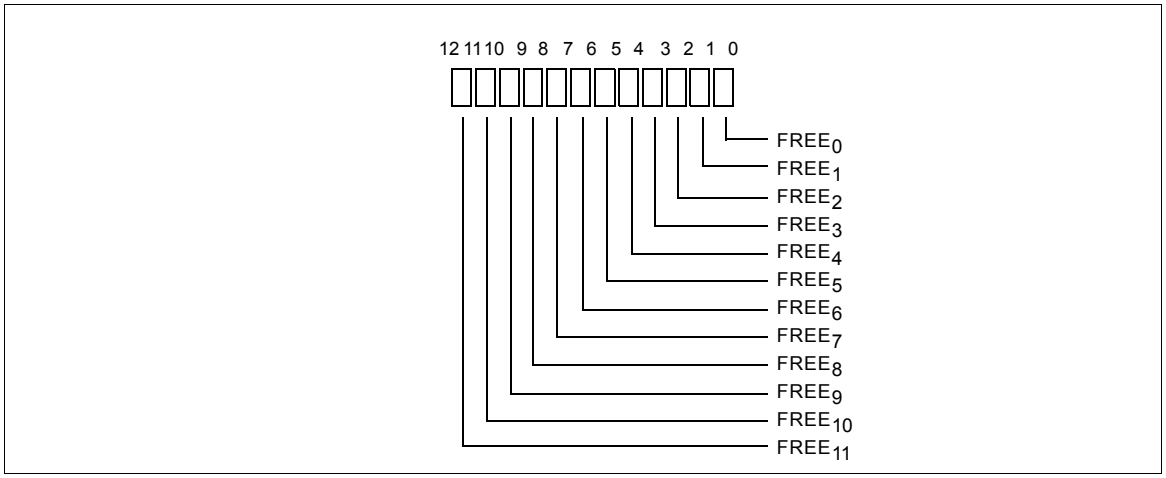

Figure 28 CNO Vector

1.7 Forming a Condition Summary and resolving the Message Engine

Each window turn will cause an activation of one of the 16 Message Engines within the TAM generator. The function of a Message Engine[1] is to apply any necessary prescales, evaluate the busy state of the LAT, and conditionally synthesize and transmit a Trigger Accept Message. Determining *which* one of the 16 engines to activate on a window turn requires two steps:

- Forming a Condition Summary using the Window Summary and the Trigger Vectors.
- Using the Condition Summary as an index to a lookup table. The entries of this table correspond to the address of a Trigger Engine.

These two steps are discussed in detail below.

1.7.1 Forming the Condition Summary

The Condition Summary is a 8-bit quantity which summarizes the triggerable conditions existing at the time a window closes. Each bit offset of a summary corresponds to a particular triggerable condition. If a field is *set*, the corresponding condition was present sometime within the duration of the trigger window. The structure of the Condition Summary is defined in Section 1.7.2. Seven of the eight fields correspond directly to the seven different conditions which could be used to open a window. That is, they are a copy of the *Window Summary* described in Section 1.6. The eighth condition uses the ROI vector (discussed in Section 1.6.2) to form coincidences either between itself, or between it and the TKR vector. The algorithm to form this condition runs only after the window has closed, as the window defines when in time coincidences can occur. (See Section 1.5.7.) This eighth condition is called the ***ROI condition***, as it is always derived using the ROI signals as input. The GEM calculates this condition two different ways, but conditionally applies only one of the two results. The two algorithms are:

TKR vetoed: In this scheme, the tiles of the ACD are used as *veto*. Each ROI corresponds to one of the of the LAT's 16 towers, and any one region is used to define which veto signals "shadow" a particular tower. A bit-wise coincidences between the ROI vector and TKR vector is formed. Any one of the 16 coincidence defines a legitimate condition. This algorithm is represented by the schematic illustrated in Figure 30.

ROI coincidence: In this scheme, the tiles of the ACD are used as a *trigger*. The 16 ROIs are divided into eight region pairs. For each region pair, one region is defined as *odd* and the other defined as *even*. Coincidences are formed between odd and even regions, allowing up to eight simultaneous coincidences to be formed. Any one of the eight coincidences defines a triggerable condition. This algorithm is represented by the schematic illustrated in Figure 31.

Which of the two algorithms used by the GEM is determined by a single-bit field of the configuration register described in Section 2.3.1. It is important to note, however, that whether

1. Explained more fully in Section 1.8.1.

or not the ROI *signal* is used to open a window (as described in Section 1.5.7) is completely independent of which algorithm the GEM uses to derive the ROI *condition*. The act of forming the Condition Summary is illustrated in Figure 29:

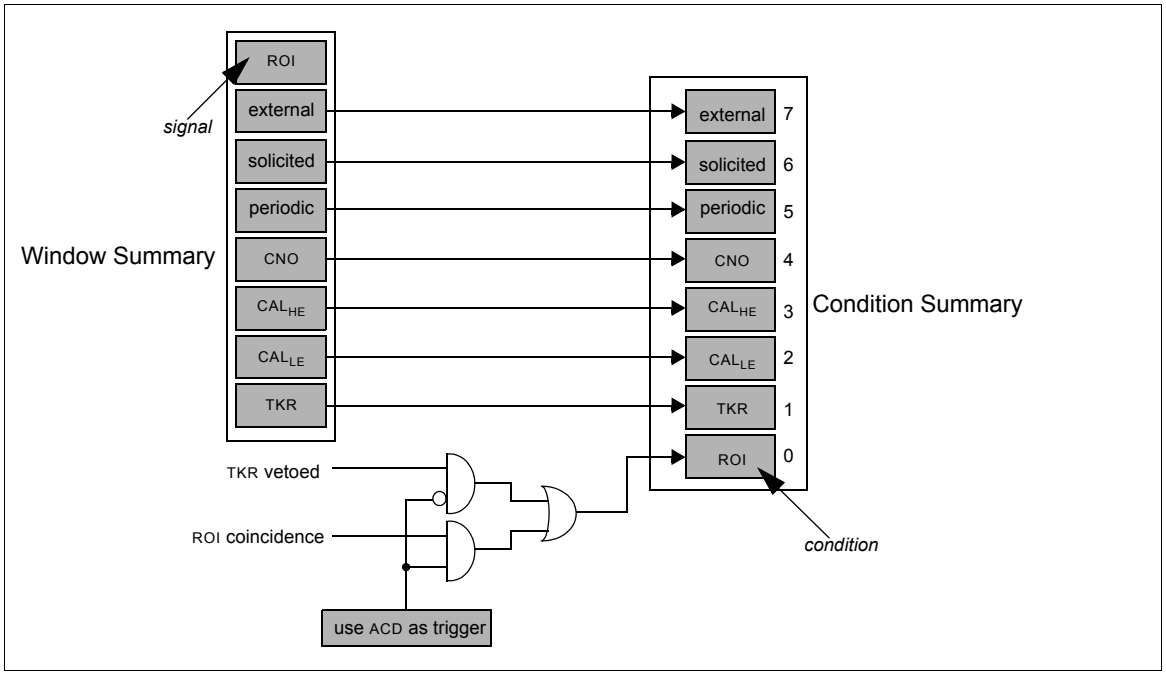

Figure 29 Forming the Condition Summary

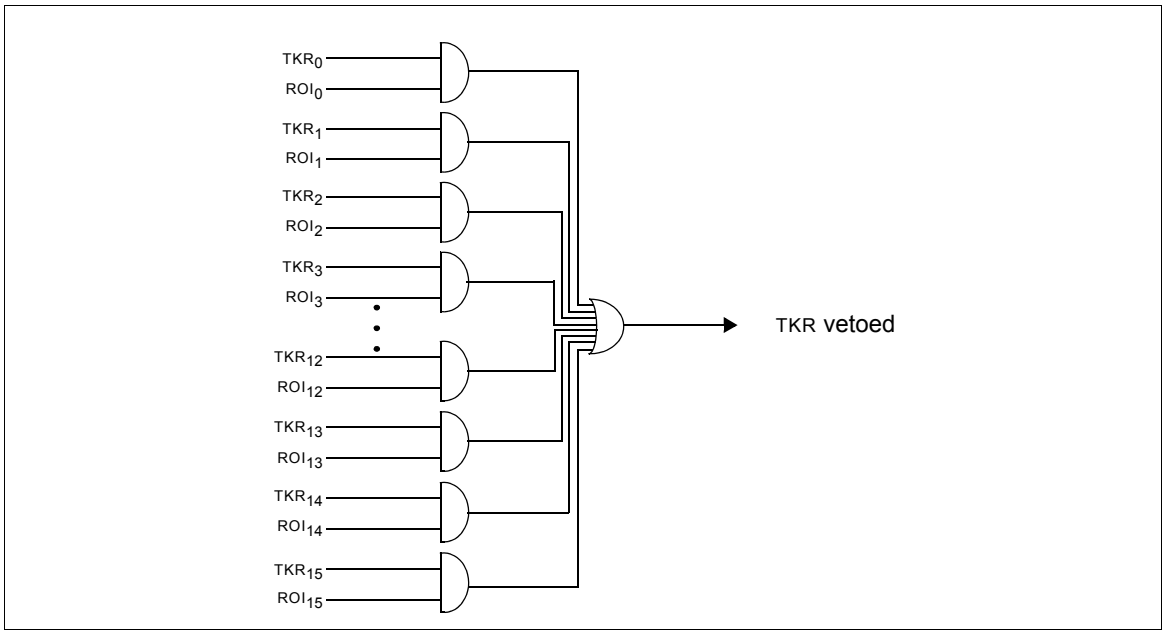

Figure 30 Forming the TKR (tracker) vetoed signal

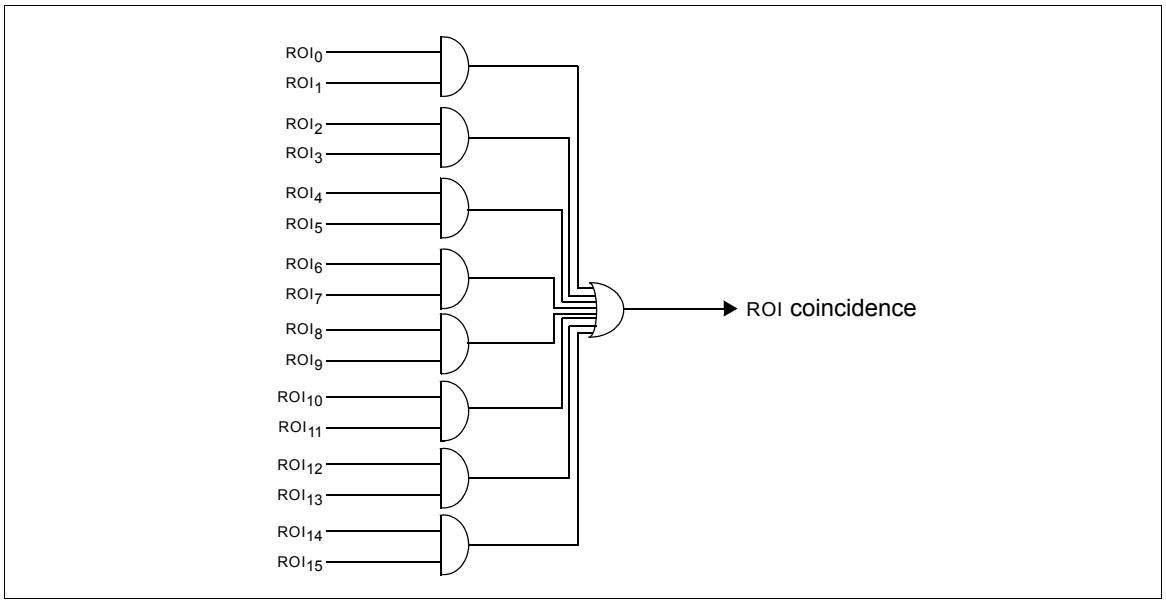

Figure 31 Forming the ROI coincidence signal

1.7.2 The Condition Summary

The format of the Condition Summary is illustrated in Figure 32. Each bit offset corresponds to one Triggerable Condition. If a bit at a particular offset is *set*, the corresponding condition was asserted for at least *one* system clock (20 MHZ) within the trigger window.

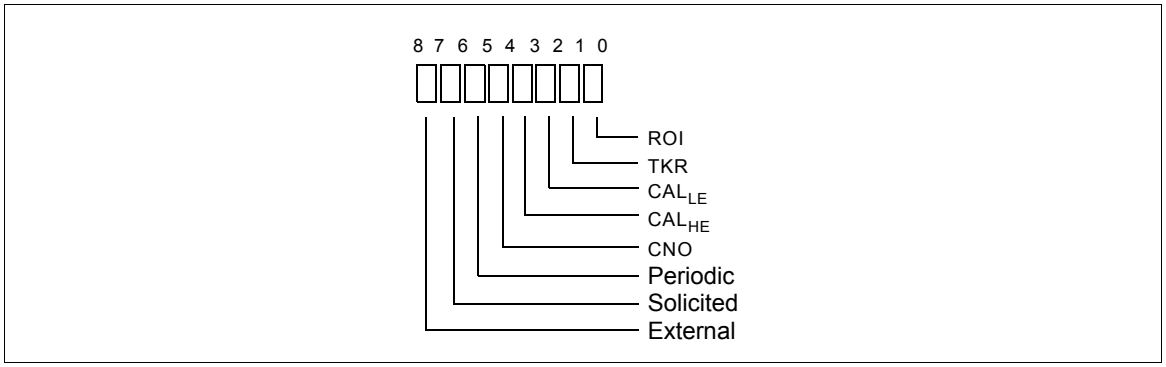

Figure 32 Condition Summary contribution to event

ROI: The interpretation of this field depends on whether the ACD was used by the GEM as a *veto* or as *trigger*. (See Section 2.3.1.) If the ACD is used a veto, this field indicates that one or more of the sixteen tower TKR signals were asserted and *at least* one of the 16 ROI signals were asserted. If the ACD is used as a trigger, this field indicates that one or more of the 8 configured coincidences between regions were found.

TKR: For each of the sixteen towers of the LAT, the tracker produces (via its TEM) one Trigger Input (the "3-in-a-row" signal). If this field is *set*, one or more of these 16 inputs were asserted.

CAL (Low Energy): For each of the sixteen towers of the LAT, the calorimeter produces (via its TEM) two Trigger Inputs, *Low* and *High* Energy. If this field is *set*, one or more of the 16 *Low* Energy inputs were asserted.

CAL (High Energy): For each of the sixteen towers of the LAT, the calorimeter produces (via its TEM) two Trigger Inputs, *Low* and *High* Energy. If this field is *set*, one or more of the 16 *High* Energy inputs were asserted.

CNO: The ACD produces twelve CNO Trigger Inputs, one from each of its FREE boards. If this field is *set*, one or more of these 12 inputs were asserted.

Periodic: The time-base of the GEM drives a periodic Trigger Input. If this field is *set*, the periodic input was asserted.

Solicited: Through its command/response interface (discussed in Section 3.3.1), the GEM allows an external user to request, or solicit, a trigger. If this field is *set*, the user *solicited* input was asserted.

External: Through a signal asserted on the external test connector of the GASU (discussed in [5]), the GEM allows an external user to request, or solicit, a trigger. If this field is *set*, the user *external* signal was asserted.

1.7.3 Looking up a Message Engine

The Scheduler, after synthesizing the condition summary, treats it as a 8-bit number and uses its value as an *index* into an internal lookup table. For example, if both the TKR and $\mathtt{CAL}_{\mathtt{HE}}$ conditions were present, the value of the Condition Summary would be 0A (Hexadecimal) and this value would also be its corresponding table index. The entries of this table correspond to the *addresses* of one of the sixteen engines on the TAM generator. As there are sixteen engines, engine addresses vary from 0 to F (Hexadecimal) and, consequently, the lookup table is 4 bits wide. The lookup table is configured through the Command/Response interface as discussed in Section 2.6.4. In this fashion, the user can map any combination of triggerable conditions into as many as sixteen different Message Engines. For example, suppose the summary described above was intended to activate Message Engine number one (1). In that case, index 0A of the lookup table would be configured with the value one (1). Pictorially, this process is illustrated by Figure 33.

In short, once a summary is established, it is used as an index to look up the appropriate Trigger Engine to further process the trigger request of the Scheduler. This processing is the topic of discussion of the following section.

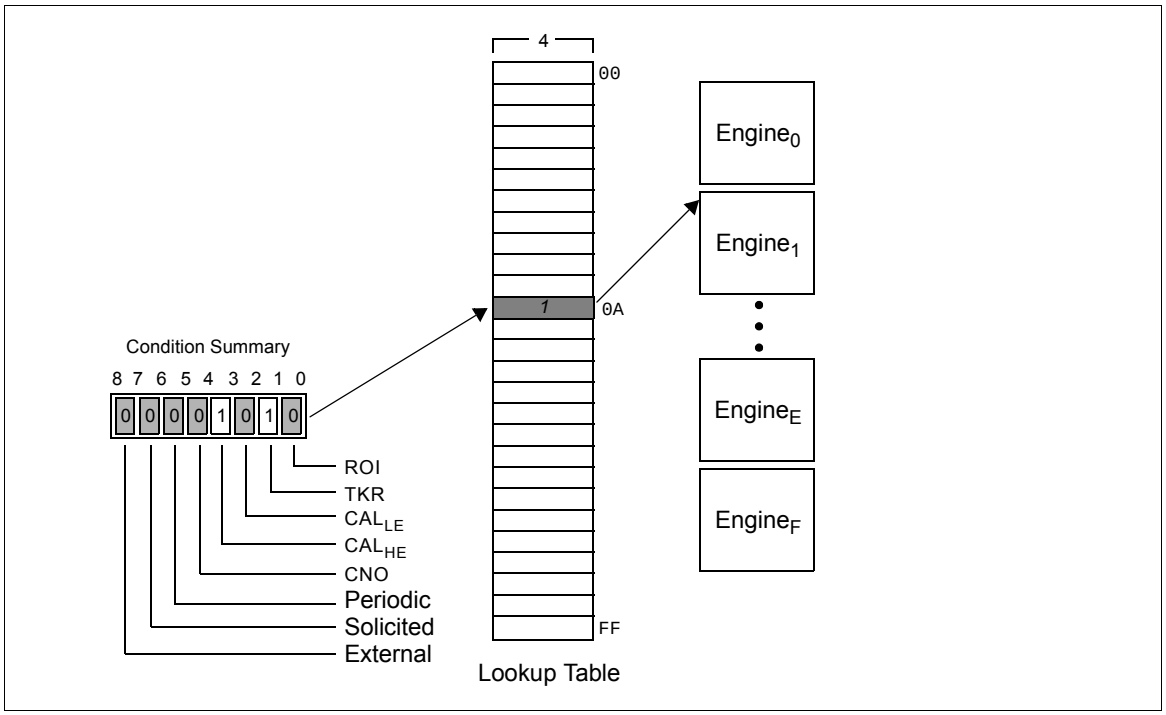

Figure 33 Resolving the Condition Summary to a Message Engine

1.7.4 The Scheduling Engine

The activity described in the previous sections can be all be abstracted as the *Scheduling Engine*. This engine is activated when either:

- A window closes. (See Section 1.5.)
- It receives a *trigger* signal. (See Section 1.8.2.)

When the engine detects a window closing it does the following:

— Samples and holds the 5 Trigger vectors

— Uses the Window Summary to construct the Condition Summary

— Uses the Condition Summary to lookup the appropriate Message Engine

— Asserts the *wakeup* signal

When the engine detects an assertion of a *trigger* signal it does the following:

— Samples and holds the 5 Trigger vectors

— Samples and holds the Condition Summary

— Samples and holds both trigger and 1-PPS times

1.8 TAM Formation

- Based on the *wakeup* and *engine select* signals, the appropriate engine is activated.
- The engine's prescaler is decremented. If expired, the engine latches its *trigger context* and asserts its *expired* signal.
- A trigger decision is made based on the *expired* signal and *busy* state of the LAT. Appropriate statistics are updated.
- The sequence counter is latched and incremented. Appropriate statistics are updated.
- The TAM is synthesized and transmitted using the latched sequence and context.
- The Event contribution is synthesized and transmitted using the latched sequence and context along with additional information latched in other blocks.

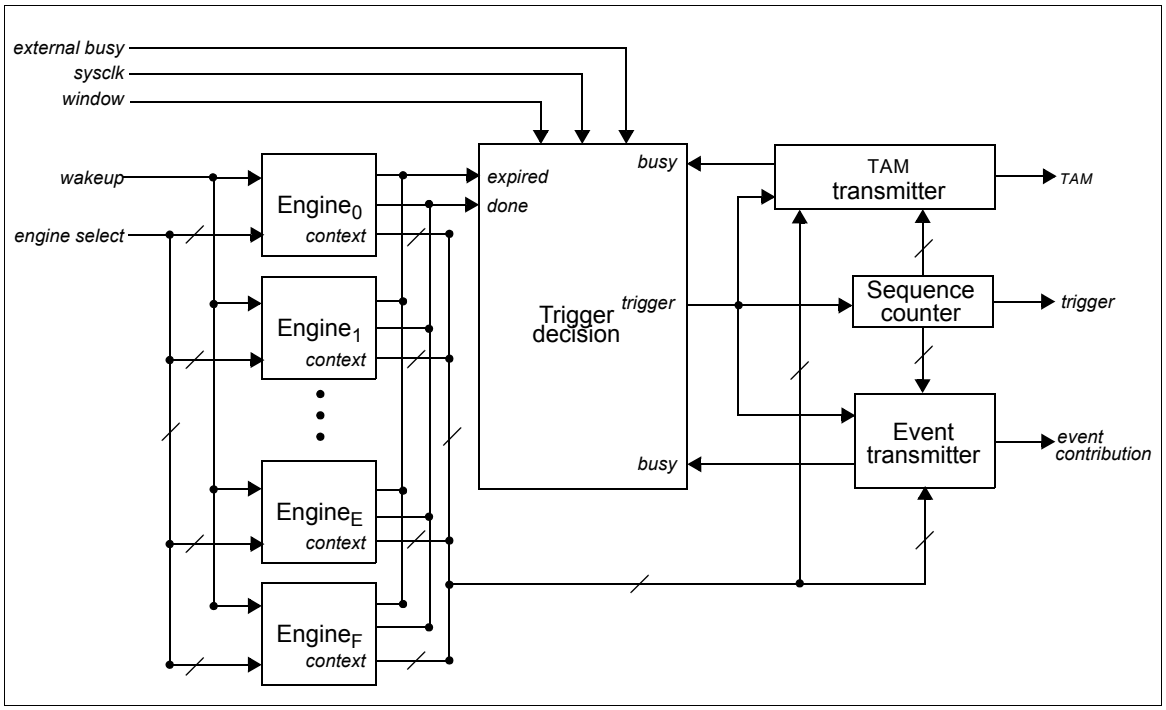

Figure 34 Block diagram of the TAM generator

1.8.1 The Message Engine

Each Message Engine contains:

- An eight-bit counter which counts *down* and includes zero-detect logic
- An inhibit flag used to mask the prescaler

- A register containing:
  - — The counter's pre-set value
  - — The inhibit flag's pre-set value
  - — The trigger context

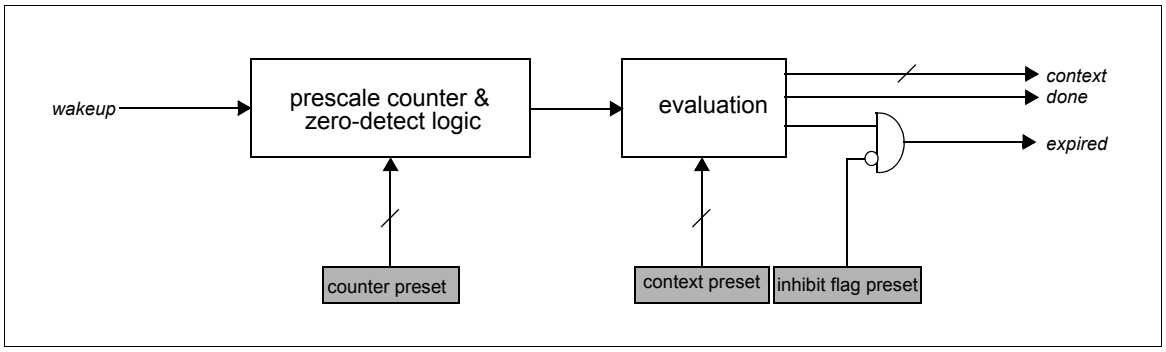

Figure 35 Block diagram of a Message Engine

The message engine is activated by the assertion of a *wakeup* signal from the Scheduler. When activated the engine determines whether the counter has a value of zero. If the counter is *not* zero, the counter is decremented and the engine asserts its *done* signal and waits for a subsequent wakeup request. If the counter is zero, the engine does the following:

— Asserts the expired signal (masked with the inhibit flag)

— Asserts its context value

— Resets both the counter and the inhibit flag using the values in the pre-set register

— Asserts its *done* signal and waits for a subsequent wakeup request

1.8.2 Making a trigger decision

This block is activated when the *done* signal is asserted by a Message Engine. (See the previous section, 1.8.1.) When active, this block has two responsibilities:

a. Generate the *trigger* signal. When this signal is asserted it will:
   - — Activate the TAM transmitter (discussed in Section 1.8.4) in order to synthesize and transmit the Trigger Accept Message
   - — Sample and hold all the quantities which will constitute the components of the GEM's event contribution.
   - — Activate the Event transmitter (discussed in Section 1.8.5) in order to synthesize and transmit the Event Contribution to the Event builder.

b. Update the appropriate statistics counters. The content of these counters are part of the quantities sampled and added to the GEM's event contribution. These counters are also available through the Command/Response interface as a set of registers as discussed in Section 2.6.

A positive trigger decision requires two conditions to be true:

a. The LAT must *not* be busy. Note that the GEM itself (through its two transmitters) may be busy processing a previous event.

b. The prescaler of the Message engine must have expired.

The functions of this block are illustrated in Figure 36:

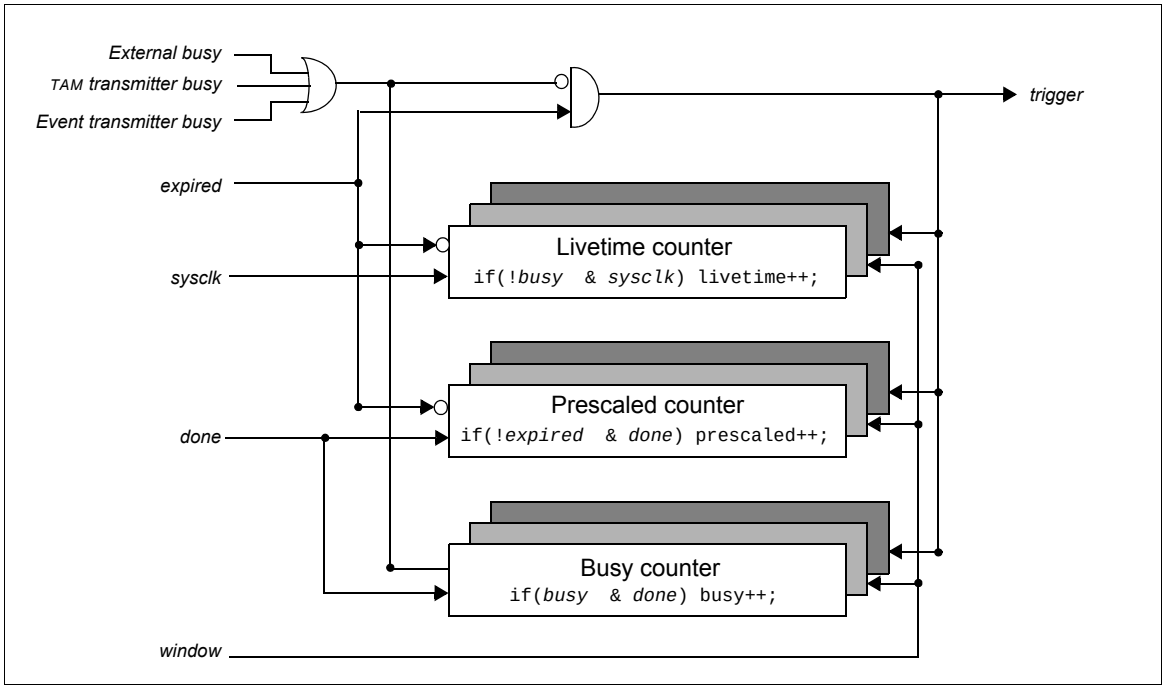

Figure 36 Making the Trigger decision

1.8.3 The Sequence Register

The GEM maintains two counters which are incremented by a *trigger* signal. The first is a 2-bit counter called the ***Tag*** counter. The second is a 15-bit counter called the ***Event Number*** counter. The initial value for both these counters is controlled through the configuration register described in Section 2.3.4. Collectively these two counters are referred to as the ***Sequence*** register. On the receipt of the *trigger* signal, the sequence register has two responsibilities:

- Latch the current value of both the ***Tag*** and ***Event Number*** counters
- Increment both the ***Tag*** and ***Event Number*** counters

These latched values are used by both the TAM and Event transmitters described below.

1.8.4 Transmitting the TAM

The receipt of the *trigger* signal activates the TAM transmitter. The transmitter assumes the current context has been latched by the appropriate Message Engine (discussed in Section 1.8.1) and the current sequence has been latched by the Sequence Counter (discussed in Section 1.8.3). Given these conditions, the transmitter performs the following actions:

- Asserts its *busy* signal.
- Using the latched context and sequence, clocks out the 32-bit TAM. (See Section 1.8.6.) The first field transmitted is a one bit field specifying the start of the message and always has a value of one (1). The last field transmitted is the parity summed over the entire message. The sign of the parity is determined by a single bit configuration field as described in Section 2.3.1. The relationship between the context and sequence and the message the transmitter generates is illustrated in Figure 38.
- Deasserts its *busy* signal.

The interfaces of the TAM transmitter are illustrated in Figure 37:

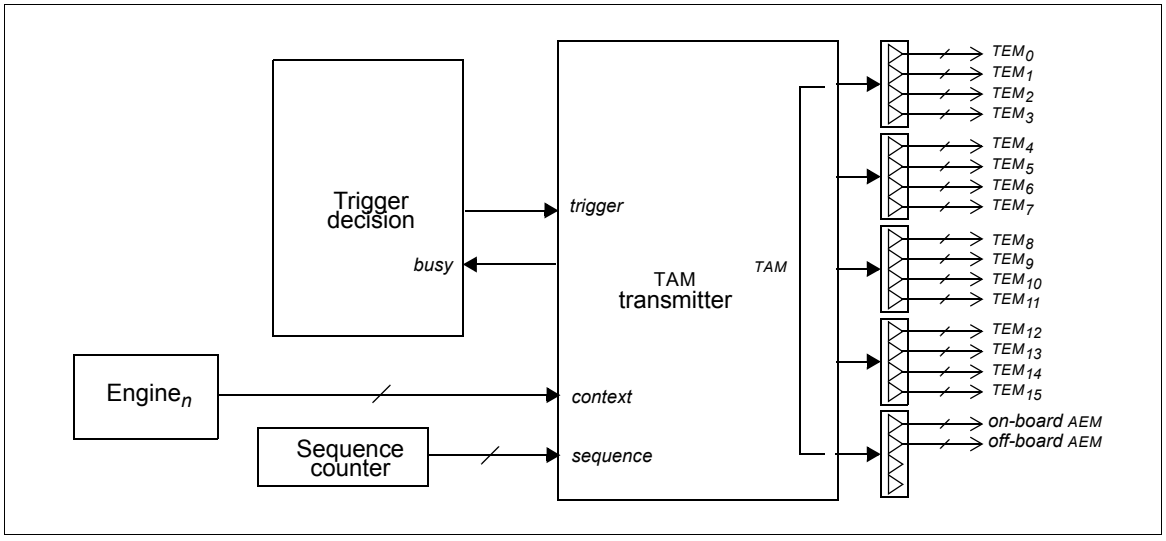

Figure 37 Block diagram of the TAM transmitter

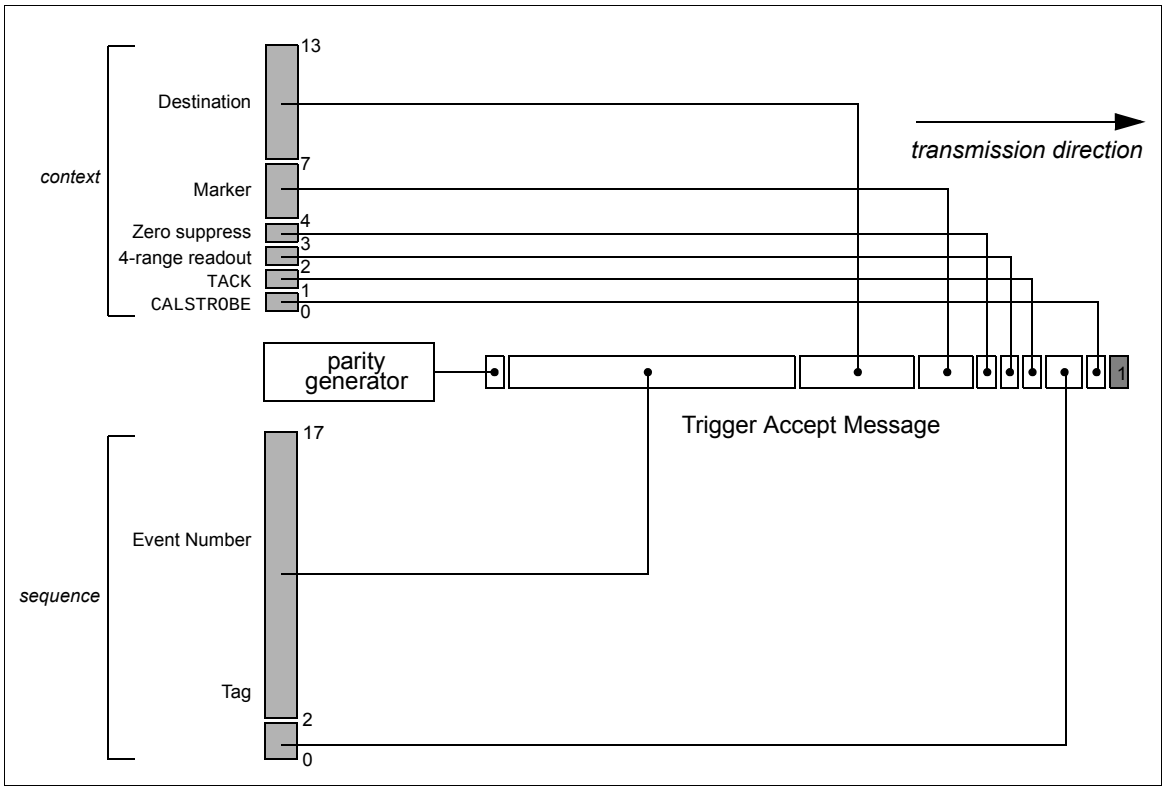

Figure 38 The TAM transmitter forming a message

1.8.5 Transmitting the Event Contribution

The receipt of the *trigger* signal activates the Event transmitter. The transmitter assumes the current context has been latched by the appropriate Message Engine (discussed in Section 1.8.1) and the current sequence has been latched by the Sequence Counter (discussed in Section 1.8.3). Given these conditions, the transmitter then performs the following actions:

- Asserts its *busy* signal.
- Updates the events sent statistics. (See 2.3.6.)
- Synthesizes and transmits one LATp packet corresponding to the GEM event contribution. The steps to build a packet include:
  - — Using both the *context* signals and the Address register (discussed in 2.3.2), synthesize and clock out the LATp header
  - — Using both the *context* and *sequence* signals, synthesize and clock out the event header
  - — For each contribution described in Chapter 4, assert the appropriate *select* signal and then clock in and clock out the corresponding contribution
- Deasserts its *busy* signal.

The interfaces of the Event transmitter are illustrated in Figure 39:

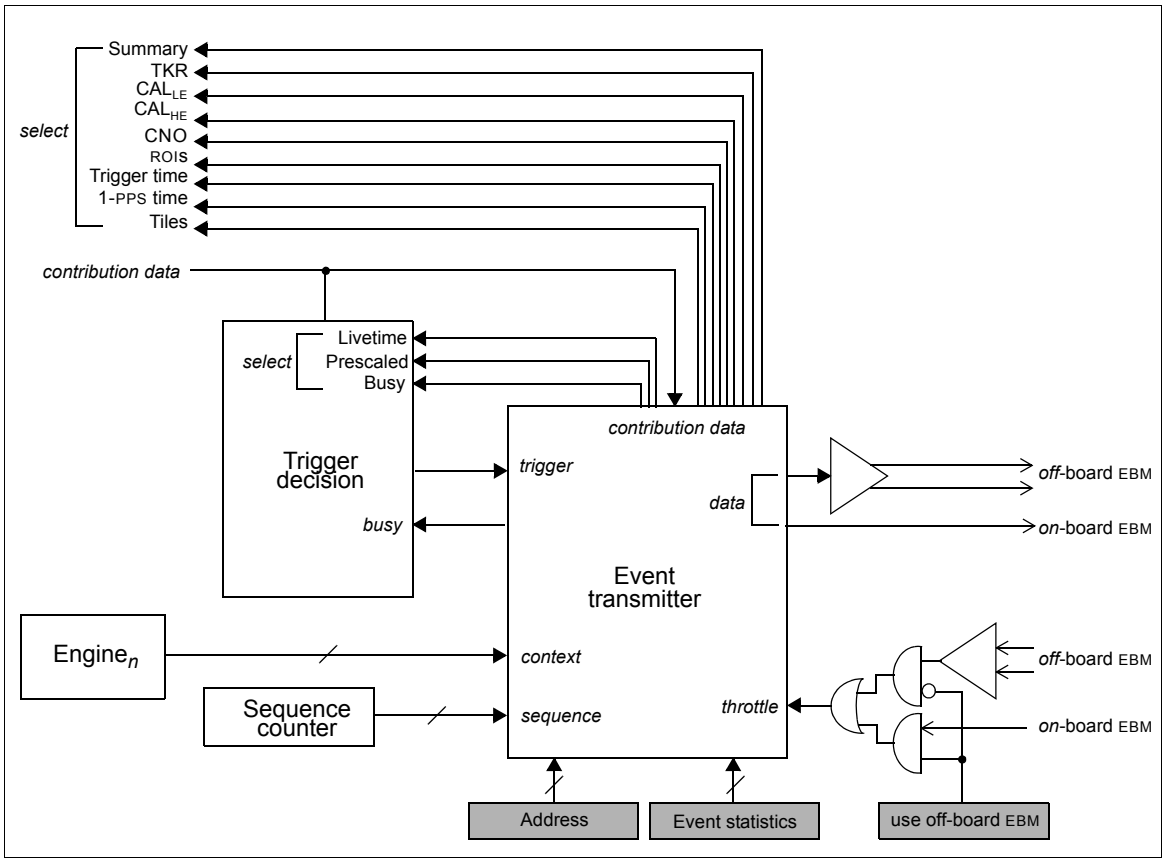

Figure 39 Block diagram of the Event transmitter

1.8.6 Trigger Accept Message structure

The message encoded and transmitted by the GEM (discussed in Section 1.8.4) whenever a positive trigger decision is reached is called the *Trigger Accept Message* (TAM). The function of this message is not only to inform its recipients that their respective systems must be read out, but also to parameterize, in a generic fashion, *how* they should be read out. The TAM is a 32-bit structure, transmitted bit serial. As these message are sent both spontaneously and unsolicited to their recipients, the first field is intended to allow a receiver to find the start of a message. The data integrity of the message is protected by a trailing 1-bit parity field. A 30-bit "payload" consisting of two components is encapsulated between these two fields, as illustrated in Figure 40:

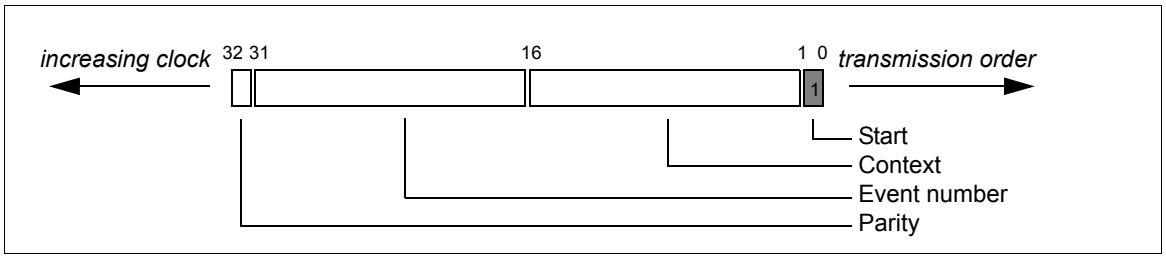

Figure 40 Structure of the Trigger Accept Message (TAM)

Start: Specifies the start of the message. This field is always *set* (1).

Context: The 15-bit Trigger context as described in the following section (1.8.6.1).

Event number: The 15 most significant bits of the 17-bit event sequence number. The low-order two bits of the sequence number are contained in the ***Tag*** field of the ***Context*** field. (See Section 1.8.6.1.)

Parity: Specifies the *odd* parity, computed over the entire 32-bit length of the message.

1.8.6.1 Trigger Context

The directions communicated by the GEM to its recipients whenever a trigger is asserted are encapsulated in that part of the TAM called the *Trigger Context*. The origin of any particular context *value* is determined by the Message Engine which produced the trigger. (See sections 1.8.1 and 2.4.) The structure of the Trigger Context field of the TAM is illustrated in Figure 41:

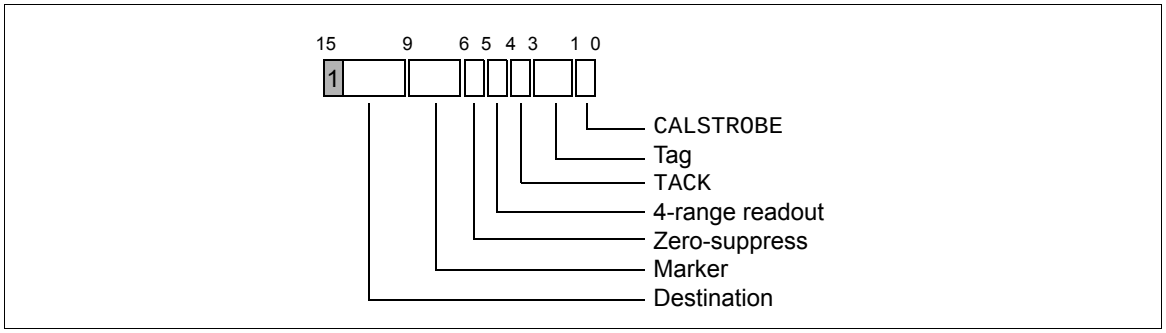

Figure 41 Structure of the Trigger Context

CALSTROBE: Specifies what type of command should be issued *first* by the recipient to its Front-End Electronics. If this field is *set*, the first command to be emitted is a CALSTROBE. If the field is *clear*, the first command to be emitted is a TACK. See Section 1.8.6.2 for a discussion on how this field is to be used.

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.

TACK: Specifies whether a *second* command should be issued by a recipient to its Front-End Electronics. If this field is *set*, a second command is to be emitted (following the first command specified in the CALSTROBE field described above). This second command is always a TACK. If the field is *clear*, a second command is *not* to be emitted. See Section 1.8.6.2 for a discussion on how this field is to be used.

4-range readout: Determines, for those recipients which have this capability,[1] whether or not their event data should include all four ranges from their digitizers. If this field is *set*, the recipient will emit all four ranges of its event data. If *clear*, the recipient will auto-range and send the one, appropriate range for its event data.

Zero suppress: Determines, for those recipients which are capable of performing zero suppression,[2] whether or not their event data should be zero suppressed. If this field is *set*, zero suppression is performed. If this field is *clear*, their event data is not zero suppressed.

Marker: User defined. This information is simply transmitted by the GEM and by convention is reflected by each recipient in their event contribution. For example, flight software will use this field in order to insert "markers" at well-known times into the event stream.

Destination: The GEM in conjunction with both its message recipients and the EBM determines the final destination address of any triggered event data. This destination is one of the potential thirty-two CPUs occupying a slot in either an EPU or SIU crate. (See [8].) Because the destination must, in all cases, be a master, the high-order bit of the address *must* be *set* (1). This field specifies the directions (or input) given by the GEM in order to determine the destination of event corresponding to the message. Table 6 enumerates how different values of this field map to possible event destinations.

Table 6 The destination field of the Trigger Accept Message

Field value[1]	Effect on event destination
0	Defer destination decision to EBM. (See [2].)
1 - 1E	Send to *one* (the specified) address.
1F	Send to *all* addresses.

1. Hexadecimal. Note: This table discusses only the five (5) low-order bits of the address.

1. The calorimeter fraction of a tower.
2. This includes both the calorimeter data from a tower and event data from the ACD.

1.8.6.2 Trigger sequencing

Among other things, the arrival of a TAM initiates the transmission, by the receiving module of combinations, of two *commands* to its corresponding Front-End electronics. These two commands are:

CALSTROBE: This command instructs the Front-End electronics to inject a fixed amount of charge at the input to their corresponding analog electronics. The amount of charge injected is considered both measurable and precise. Typically, the amount of charge is programmable through one or more of the registers of the Front-Ends. The principal use of CALSTROBE command is as a mechanism to perform electronics calibrations.

TACK: This command instructs the Front-End electronics to latch and hold the input to their corresponding analog electronics. This held information forms the basis for the module's event contribution. The principal use of TACK command is as a mechanism to perform event readout.

Each subsystem provides its own individual implementation of each command. The structure of these commands is described in Section 1.8.6.3. Once a TAM has been received by a module (the TEM or AEM), it must synthesize a trigger sequence composed of one or two of these commands. There are three possible sequences:

i. A single CALSTROBE command

ii. A single TACK command

iii. A CALSTROBE followed by a TACK command with a fixed (but programmable) interval between the two commands.

The CALSTROBE and TACK fields of the Trigger context determine how a module reacts to receiving a TAM, as enumerated in Table 7:

Table 7 Response of a module to receiving a Trigger Accept Message

CALSTROBE	TACK	Module response
True	*False*	Only a CALSTROBE command is transmitted to the Front-End electronics. The module will not emit an event.
False	*Don't Care*	Only a TACK command is transmitted to the Front-End electronics. The CALSTROBE field of the message is ignored. The module will emit *one* event.
True	*True*	A CALSTROBE command followed by a TACK command is to be transmitted to the Front-End electronics.The timing between these two different commands is specified by the module itself.[1] The module will emit *one* event.

1. Typically, by a configuration register of the module. (See below.)

Sequence *timing* depends on whether the sequence has one or two commands. It also depends on whether the TAM is processed by TEM or AEM. For the TEM, a sequence must be distributed

to both Calorimeter and Tracker Front-End electronics. For the first, *one* command case, the sequence timing on the TEM is illustrated in Figure 42:

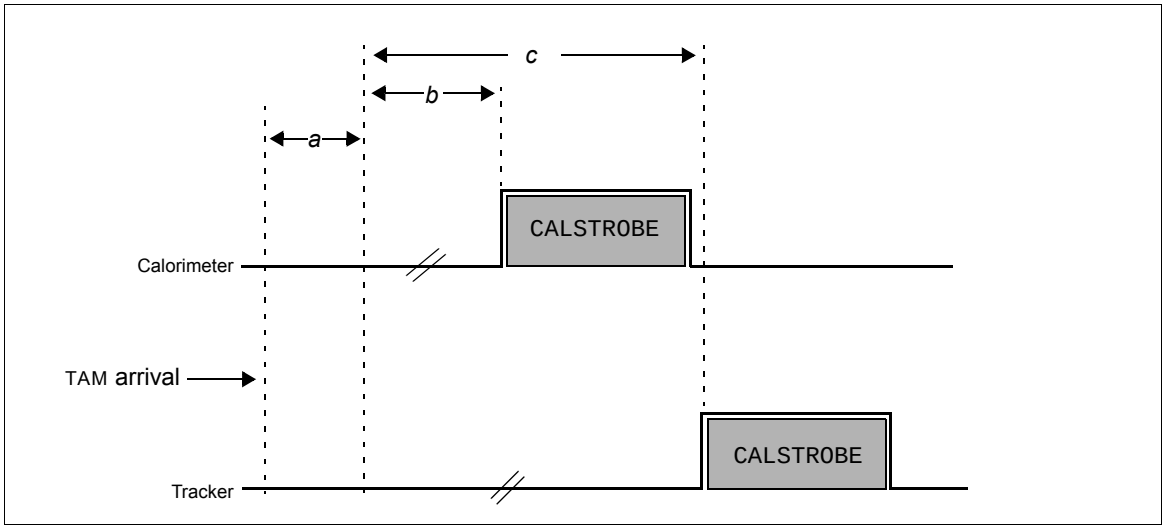

Figure 42 TEM trigger Sequence timing for CALSTROBE commands

Where:

***a*:** Is a fixed (5 system clocks?) delay imposed by the TEM in order to decode the TAM and both select and synthesize the appropriate commands.

***b*:** Is a programmable delay (determined by the CALSTROBE delay field of the *calorimeter* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the one calorimeter command, which in this case is a CALSTROBE.

***c*:** Is a programmable delay (determined by the CALSTROBE delay field of the *tracker* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the one tracker command, which in this case is a CALSTROBE.

For the second, *one* command case, the sequence timing on the TEM is illustrated in Figure 43:

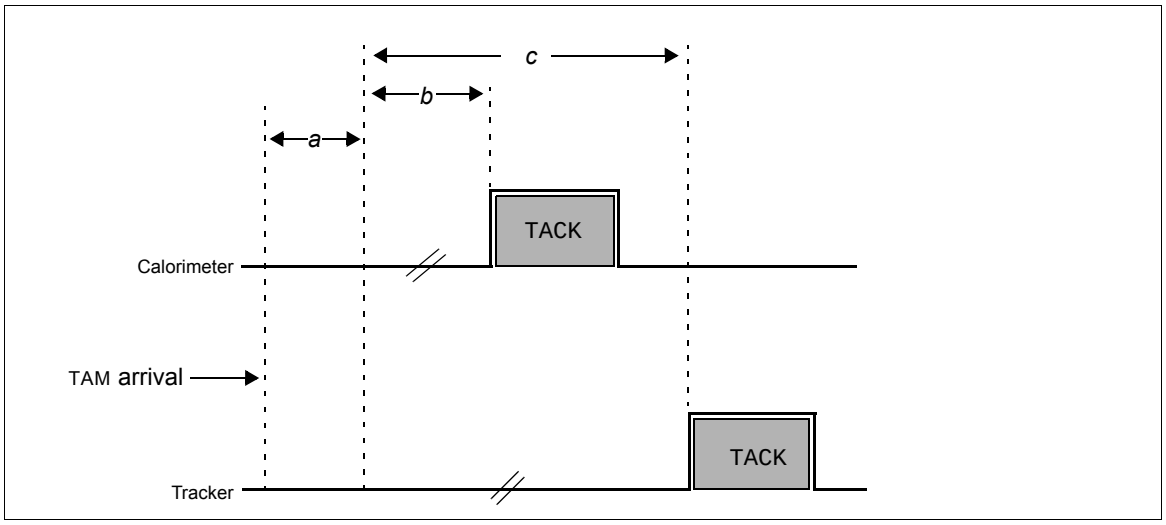

Figure 43 TEM trigger Sequence timing for TACK commands

Where:

***a*:** Is a fixed (5 system clocks?) delay imposed by the TEM in order to decode the TAM and both select and synthesize the appropriate commands.

***b*:** Is a programmable delay (determined by the TACK delay field of the *calorimeter* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the one calorimeter command, which in this case is a TACK.

***c*:** Is a programmable delay (determined by the TACK delay field of the *tracker* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the one tracker command, which in this case is a TACK.

For the *two* command case, the sequence timing on the TEM is illustrated in Figure 44:

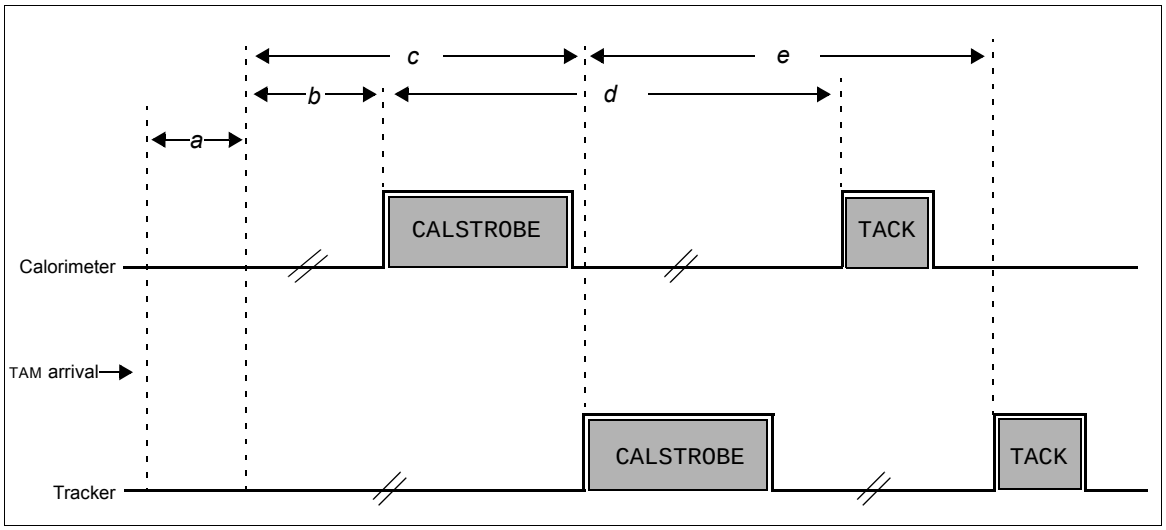

Figure 44 TEM trigger sequence timing for two commands

Where:

***a*:** Is a fixed (5 system clocks?) delay imposed by the TEM in order to decode the TAM and both select and synthesize the appropriate commands.

***b*:** Is a programmable delay (determined by the CALSTROBE delay field of the *calorimeter* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the first calorimeter command, which in this case must be a CALSTROBE.

***c*:** Is a programmable delay (determined by the CALSTROBE delay field of the *tracker* trigger sequencing register, discussed in [6]) until the *start* of the transmission of the first tracker command, which in this case must be a CALSTROBE.

***d*:** Is a programmable delay (determined by the TACK delay field of the *calorimeter* trigger sequencing register, discussed in [6]) from the *start* of transmission of the first calorimeter command, until the *start* of transmission of the second command, which in this case must be a TACK.

***e*:** Is a programmable delay (determined by the TACK delay field of the *tracker* trigger sequencing register, discussed in [6]) from the *start* of transmission of the first tracker command, until the *start* of transmission of the second command, which in this case must be a TACK.

NOTE: AEM timing remains to be written.

1.8.6.3 Trigger Command structure by subsystem

Note: This section is not correct or complete.

The TEM must synthesize two variants of the two different trigger commands, one for the calorimeter and the other for the tracker. The structure of these four commands are given in the figures below:

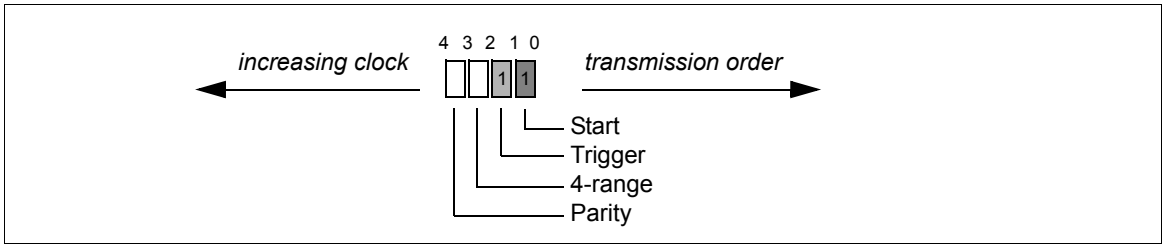

Figure 45 Structure of the TACK command for the Calorimeter electronics

Start: Specifies the start of the command. This field is always *set* (1).

Trigger: Specifies whether the command is Trigger or Register related. Trigger requests and Register Read/Writes share the same physical wire. This field is used by the electronics to differentiate the type of command. Thus, for Trigger requests, this field is always *set* (1).

4-range readout: Determines whether or not calorimeter data should include all of its four ranges. If this field is *set*, the recipient will emit all four ranges. If *clear*, the electronics will auto-range and send the one, appropriate range. The value for this field is derived from the Trigger Context of the TAM. (See Section 1.8.6.1.)

Parity: Specifies the *odd* parity, computed over fields *1* and *2* of the command.

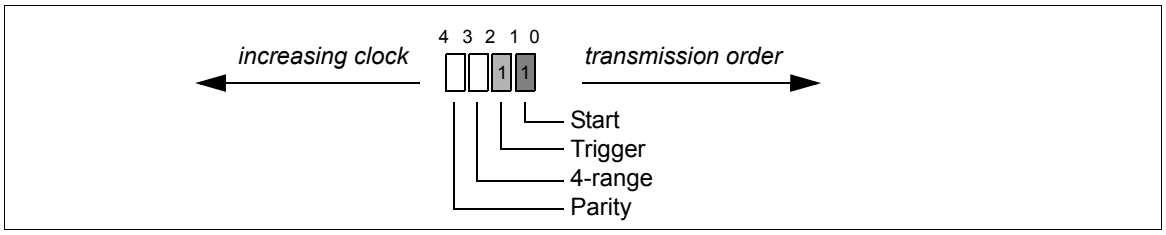

Figure 46 Structure of the CALSTROBE command for the Calorimeter electronics

Start: Specifies the start of the command. This field is always *set* (1).

Parity: Specifies the *odd* parity, computed over fields *1* and *2* of the command.

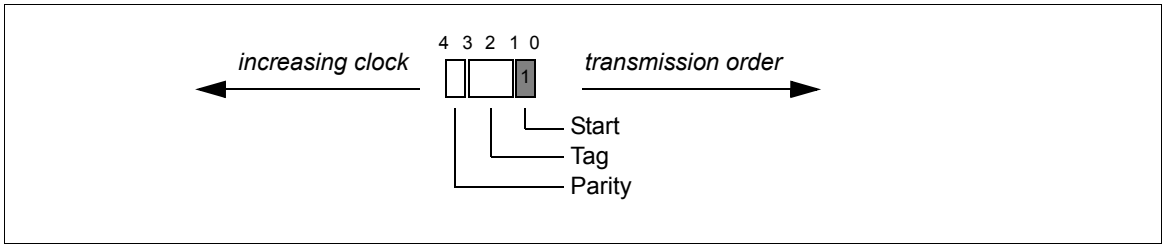

Figure 47 Structure of the TACK command for the Tracker electronics

Start: Specifies the start of the command. This field is always *set* (1).

Tag: The value for this field is derived from the Trigger Context of the TAM (discussed in Section 1.8.6.1) and represent the two *least* significant bits of the 17-bit event sequence number.

Parity: Specifies the *odd* parity, computed over fields *xxx* of the command.

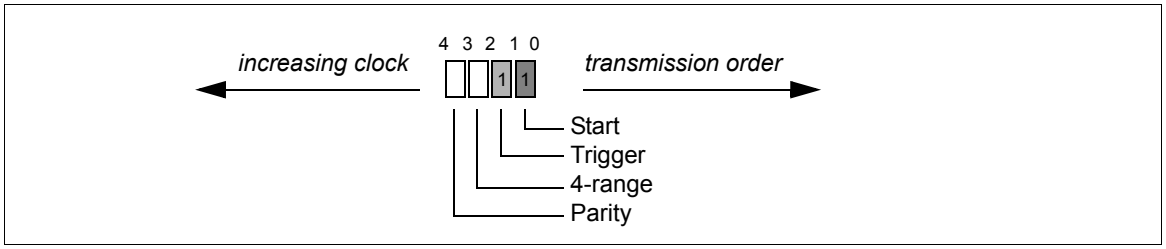

Figure 48 Structure of the CALSTROBE command for the Tracker electronics

Start: Specifies the start of the command. This field is always *set* (1).

Parity: Specifies the *odd* parity, computed over fields *1* and *2* of the command.

For the ACD, all commands go through the AEM and the structure of these commands is:

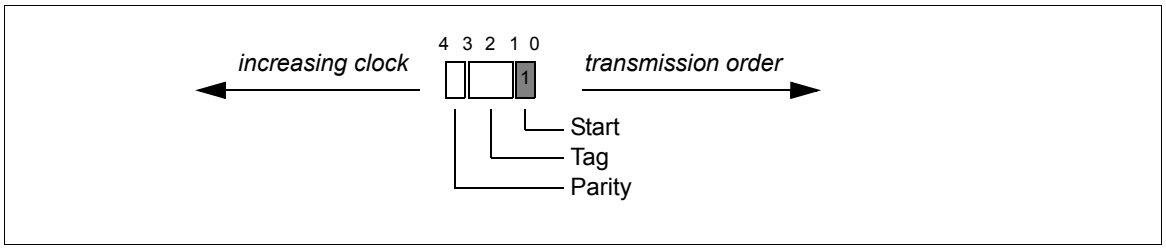

Figure 49 Structure of the TACK command for the ACD electronics

Start: Specifies the start of the command. This field is always *set* (1).

Tag: The value for this field is derived from the Trigger Context of the TAM (discussed in Section 1.8.6.1) and represent the two *least* significant bits of the 17-bit event sequence number.

Parity: Specifies the *odd* parity, computed over fields *xxx* of the command.

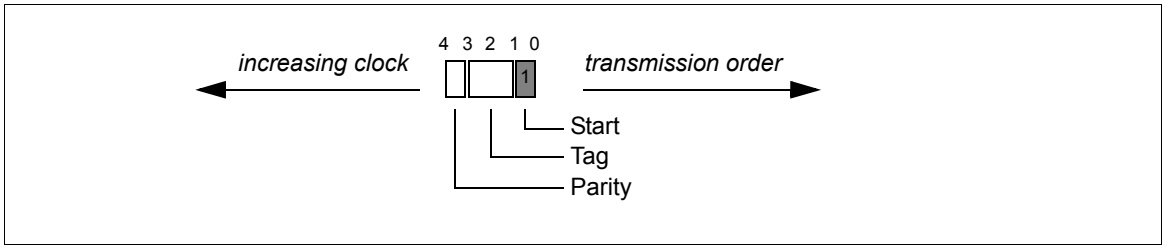

Figure 50 Structure of the CALSTROBE command for the ACD electronics

Start: Specifies the start of the command. This field is always *set* (1).

Tag: The value for this field is derived from the Trigger Context of the TAM (discussed in Section 1.8.6.1) and represent the two *least* significant bits of the 17-bit event sequence number.

Parity: Specifies the *odd* parity, computed over fields *xxx* of the command.

1.9 The Timebase

To be written.

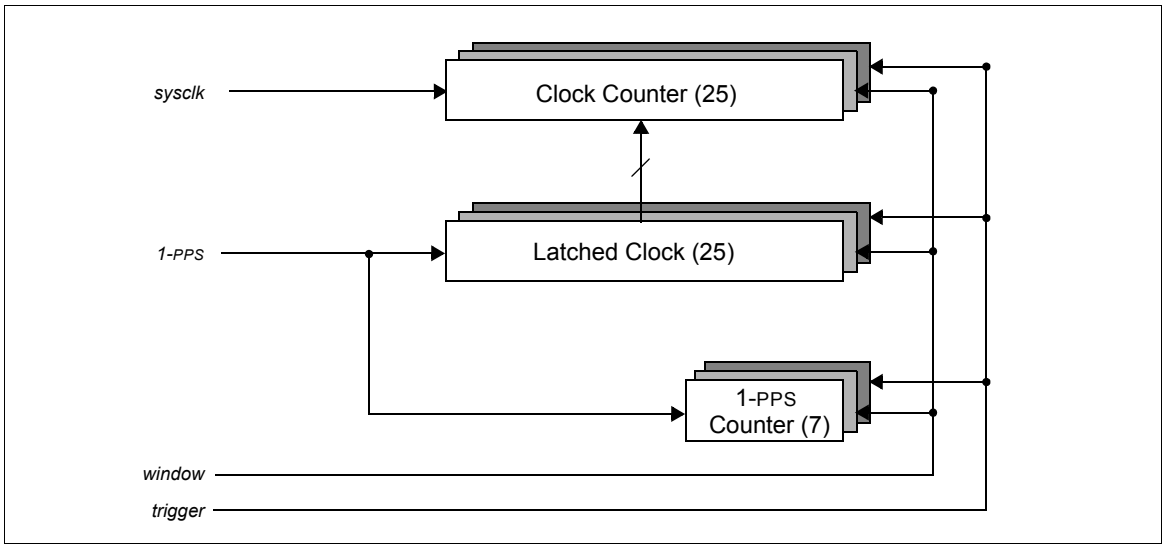

Figure 51 Block diagram of the latch and count block of the Timebase

1.10 Veto Sample and Hold

To be written.

1.11 Trigger System latency and timing

To be written.

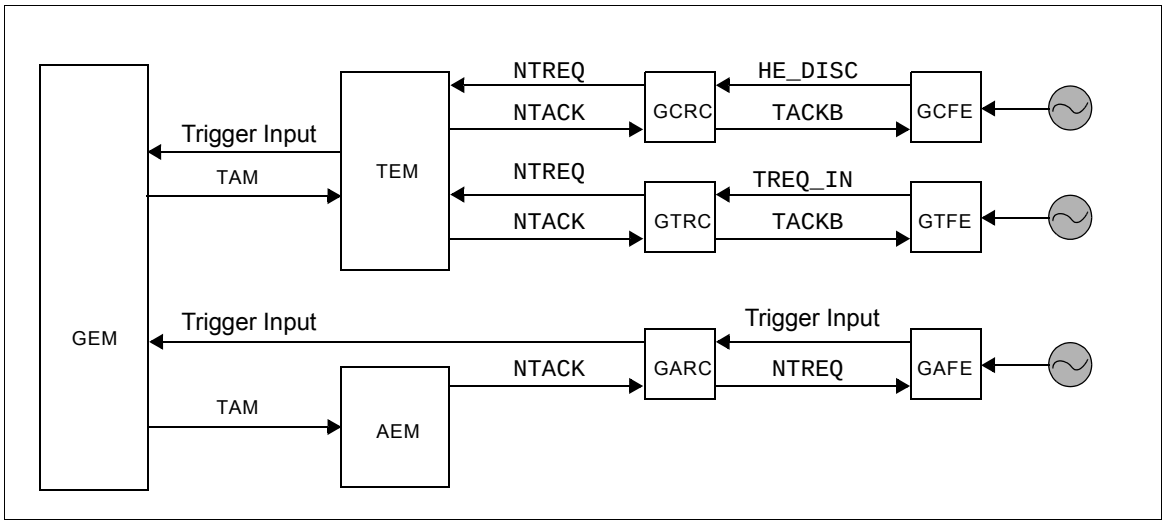

Figure 52 Round-trip between Trigger Inputs and resulting Trigger

Chapter 2
Registers

2.1 Introduction

The GEM contains a series of *registers* for configuration and management. These registers are divided into six groups of related functionality:

- GEM Control
- Window
- The Trigger Accept Message (TAM) Generator
- Trigger Statistics
- The Scheduler
- The Region Of Interest (ROI) Generator
- Input Enable

Within each group, registers have a common length to allow their access through broadcast operations. The access model for these registers is through the LAT common Command/Response Protocol. This protocol is specified generically in [1]. The specific implementation of this protocol for the GEM is described in Chapter 3.

2.2 Conventions

Certain conventions apply to the fields within a register. These conventions fit into one of three classes:

Not defined: Undefined fields are identified as Must Be Zero (MBZ) and are illustrated *grayed 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* grayed-out along with their intended 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 zero (0) corresponds to a register's Least Significant Bit (LSB), and thirty-one (decimal) corresponds to a register's Most Significant Bit (MSB). Register addresses are specified in *hexadecimal* unless otherwise noted.

2.3 GEM controller registers

This section incorporates all the registers whose configuration has a global affect over all the functional blocks of the GEM.

Table 8 The GEM control registers

Name	Address	Access	Description
CONFIGURATION	00	R/W	Configuration and setup
ADDRESS	01	R/W	LATp node address
RESERVED	02	R/W	Not used (used to be WINDOW_OPEN_MASK)
PERIODIC_MODE	03	R/W	Set bounded/free-run mode for periodic input
PERIODIC_ LIMITS	04	R/W	Set boundaries on periodic trigger input
PERIODIC_RATE	05	R/W	Sets the rate of the *periodic* trigger source
SEQUENCE	06	R/W	Current event number and tag
C/R_STATISTICS	07	R/W[1]	Command/response statistics
EVENT_STATISTICS	08	R/W[1]	Event statistics
DELAY_EXT_TRG	09	R/W	Programmable delay of external trigger condition
Total	**10**		

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

2.3.1 Configuration register

In general, this register allows defeating those features of the GEM which in the normal course of operation would always be enabled. This functionality is present only to allow *testing* of those features and perhaps to recover from single-point failure. Great care should be exercised in using any other than the default values for this register.

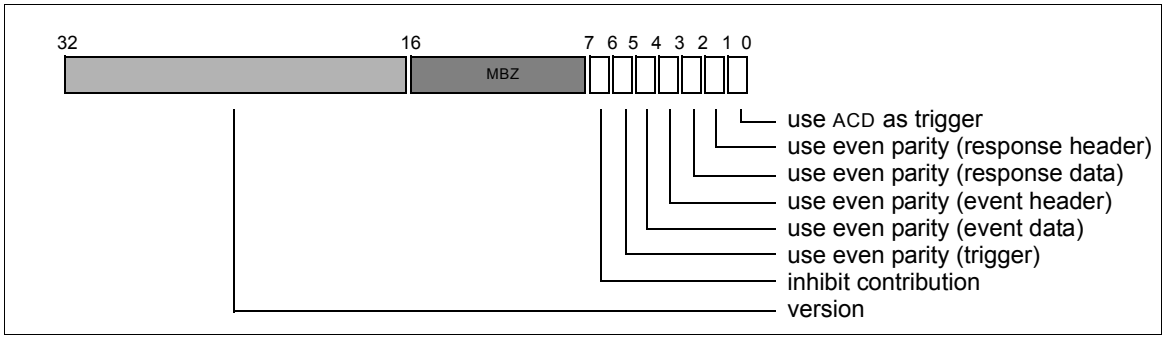

Figure 53 Configuration register

use ACD as trigger: Determines whether the GEM uses the ACD trigger input signals as a veto or as a *trigger*. If the field is *clear*, the GEM uses the ACD signals as a *veto*. If the field is *set*, the GEM uses the ACD signals as a *trigger*. (See Section 1.3.1 and Section 2.8).

use even parity (response header): Determines whether packet *header* parity generated by the GEM when transmitting response packets is *odd* or *even*. If the field is *clear*, *odd* header parity is generated. If the field is *set*, *even* header parity is generated. Note: This field is intended to be used to test whether the response receiver will in fact detect parity errors. In normal operation this field should be always be *clear*.

use even parity (response data): Determines whether packet *cell* parity generated by the GEM when transmitting response packets is *odd* or *even*. If the field is *clear*, *odd* cell parity is generated. If the field is *set*, *even* cell parity is generated. Note: This field is intended to be used to test whether the response receiver will in fact detect parity errors. In normal operation this field should be always be *clear*.

use even parity (event header): Determines whether packet *header* parity generated by the GEM when transmitting event packets is *odd* or *even*. If the field is *clear*, *odd* header parity is generated. If the field is *set*, *even* header parity is generated. Note: This field is intended to be used to test whether an event receiver will in fact detect parity errors. In normal operation this field should be always be *clear*.

use even parity (event data): Determines whether packet *cell* parity generated by the GEM when transmitting event packets is *odd* or *even*. If the field is *clear*, *odd* cell parity is generated. If the field is *set*, *even* cell parity is generated. Note: This field is intended to be used to test whether an event receiver will in fact detect parity errors. In normal operation this field should be always be *clear*.

use even parity (trigger): Determines whether the parity generated by the GEM when transmitting TAMs is *odd* or *even*. If the field is *clear*, *odd* parity is generated. If the field is *set*, *even* parity is generated. Note: This field is intended to be used to test whether a TAM receiver will in fact detect parity errors. In normal operation this field should be always be *clear*.

inhibit contribution: Determines whether or not the GEM transmits its own event contribution whenever it declares an event. If the field is *clear*, a contribution is transmitted. If the field is *set*, a contribution is *not* sent. Note that inhibiting the GEM's event contribution also eliminates the deadtime associated with sending the contribution. The principal function of this field is to test the Event Builder's reaction when it does not receive a GEM contribution, which implies that this a test feature only.

version: Specifies the hardware revision level of the GEM. The structure of this field is defined in Figure 54. Note that this field is *read-only*.

2.3.1.1 Version ID

The fields of this register are somewhat self-explanatory with the exception of the ***type*** field. This field is intended to differentiate both the context in which module was implemented and how it was intended to be used. The values for this field are defined in Table 9.

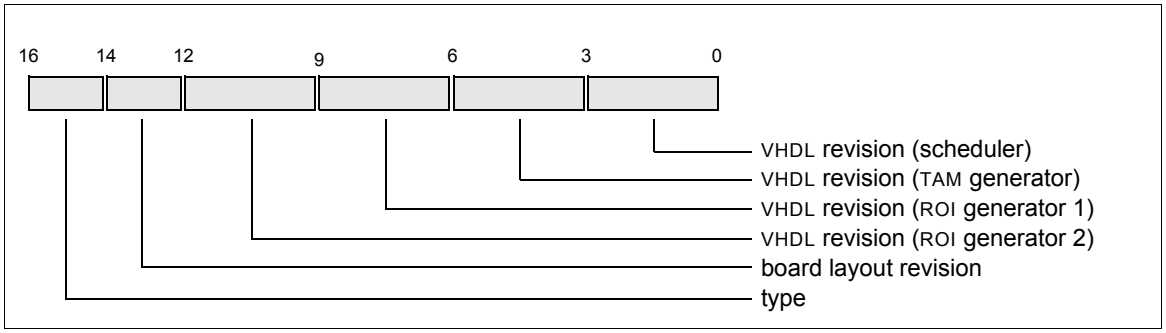

Figure 54 Structure of Revision Register or field

Table 9 Usage of the type field of the revision register

Value[1]	Description
00	Software emulation
01	Engineering model
10	Qualification model
11	Flight model

1. In binary.

2.3.2 Address register

This register is used to specify the GEM's node address on the Command/Response fabric. (See [1].) In addition, this value is also used by the GEM to set the source address in the LATp header of all event packets sent by it in response to a trigger. Note that all nodes on any one fabric must have a unique value. This register allows for definition of *only* the lower five bits of the address. As the GEM is a slave on both Command/Response and Event Fabrics, the high order bit is an implied *zero* (0).

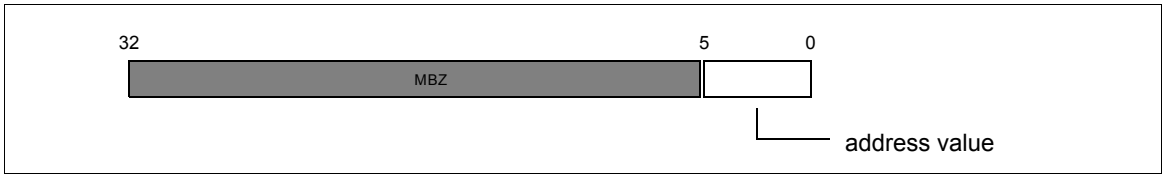

Figure 55 Address register

2.3.3 Controlling the periodic trigger input

The GEM generates a periodic signal which is used as one of the seven signals capable of opening the trigger window. (See Section 1.5.) This signal is called the ***periodic*** trigger input.

2.3.3.1 Periodic Trigger Rate register

This register has two functions:

a. Selects which of two sources are used to derive the periodic trigger. These are:
   — The system clock (`sysclk`), operating at a nominal rate of 20 MHZ
   — The 1-PPS signal, operating at (you guessed it) 1 HZ
b. Determines the rate of the periodic trigger. This register is used to pre-scale the source rate.

The "prescale" field of the register described in Figure 56 contains the pre-scaler. The pre-scaler has a range of twenty-four-bits. A value of *zero* is not allowed and setting the pre-scaler to this value will give unpredictable results. A value of one (1) divides the input rate by two (2), a value of two (2) divides the input rate by three (3), and so forth. Consequently, the largest pre-scale which can be applied is 2^{24} - 1 + 1, or 167,772,216(decimal). Which source is used is determined by the "use 1-PPS field" of the register. If this field is *clear*, `sysclk` is used as the source. If this field is *set*, the 1-PPS signal is used as the source.

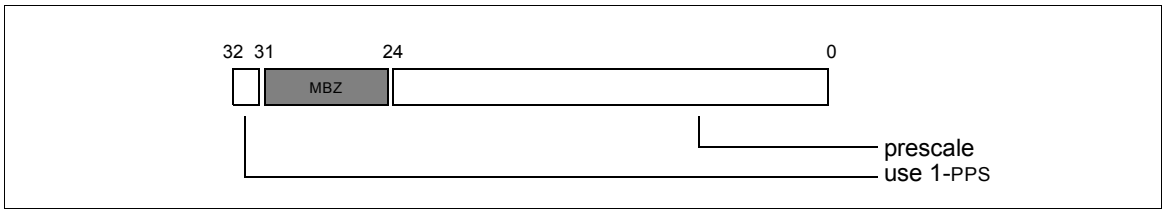

Figure 56 The Periodic Trigger Rate register

2.3.3.2 Periodic mode register

This register specifies whether the GEM will deliver the periodic trigger as either a:

- Continuous, uninterrupted number of pulses. This mode is called “free-run.”
- Fixed, limited number of pulses. This mode is called “limiting.”

This register has a single bit field to control the mode, as illustrated in Figure 57. If the field is *clear*, the GEM is in *limiting* mode. If the field is *set*, the GEM is in *free-run* mode. In free-run mode, the register described in Section 2.3.3.3 is ignored. In limiting mode, the register described in Section 2.3.3.3 determines how many pulses should be delivered before the periodic trigger is disabled.

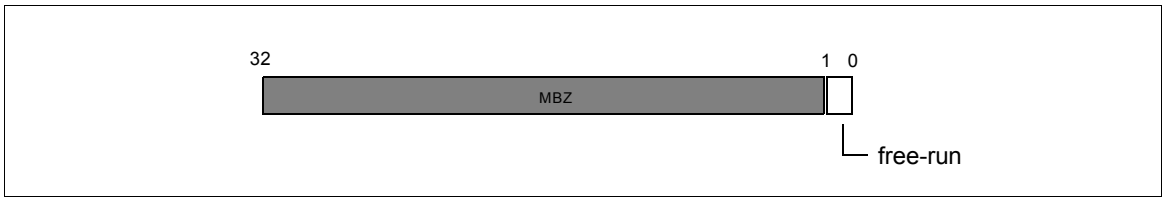

Figure 57 The periodic mode register

2.3.3.3 Periodic Limit register

This register is only applicable if the GEM is in “limit” mode. (See Section 2.3.3.2.)

This register determines the number of periodic pulses delivered by the GEM. Behind this register is a count-down counter. (See Section 1.5.6.) As the GEM generates periodic inputs, this register counts down until it reaches *zero* and the GEM then disables periodic triggers. Consequently, the value returned by reading this register indicates how many triggers remain before the periodic trigger is disabled. Each time the register is written, the counter is reset. For example, writing *zero* to this register will always abort whatever countdown sequence is in progress. As this is a sixteen-bit counter, the largest number of pulses which can be programmed is 2^{16} - 1, or 65,536 (decimal).

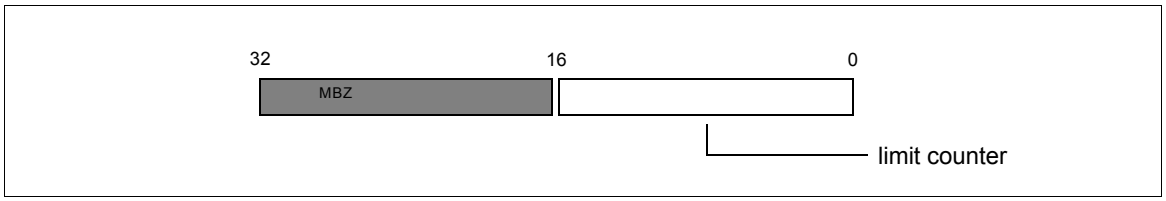

Figure 58 The Limit register

2.3.4 Sequence register

The message encoded and transmitted by the GEM when it declares a trigger is called the *Trigger Accept Message* (TAM). The structure and content of this message is described in Section 1.8.6. User control of message content is managed through two entities:

- The sequence register, which is described here.
- The *sixteen Message Engines* of the TAM generator, which are discussed in Section 2.4.

For each TAM sent, the GEM tags the message with a sequence number. Once a message has been sent, the GEM will "increment" this sequence number. This register determines both the initial and current value of the sequence number and is illustrated in Figure 59:

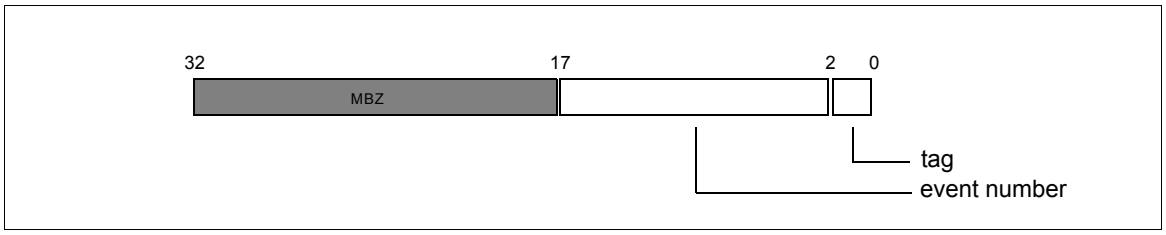

Figure 59 The sequence register

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.

event number: The 15 most significant bits of the 17-bit event sequence number. The low-order two bits of the sequence number are contained in the ***tag*** field.

2.3.5 Command/Response statistics register

The GEM is a node on the Command/Response fabric. As such it is obligated (as discussed in [1]) to keep statistics on both the commands it receives and the responses it transmits. These statistics are accessed in the register illustrated in Figure 60. Note that when the register is written, the value to be written will be ignored and the register set to *zero*.

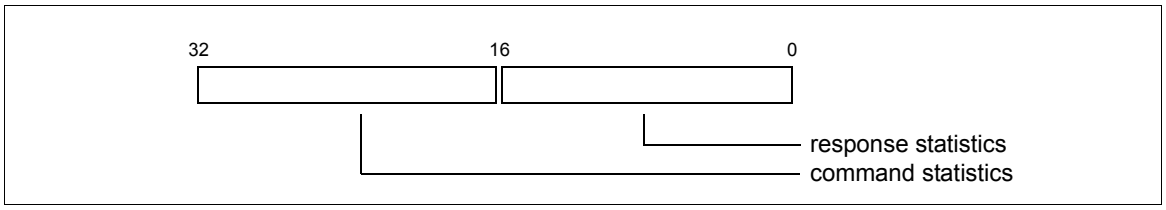

Figure 60 The Command/Response statistics register

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.

2.3.6 Event statistics register

In addition to generating trigger, the GEM also contributes data to the overall event. (See Chapter 4.) Event contributions are sent by the GEM to the EBM as LATp packets. This register contains the packet statistics related to event transmission.

Note: When the register is written, the value to be written will be ignored and the register set to zero.

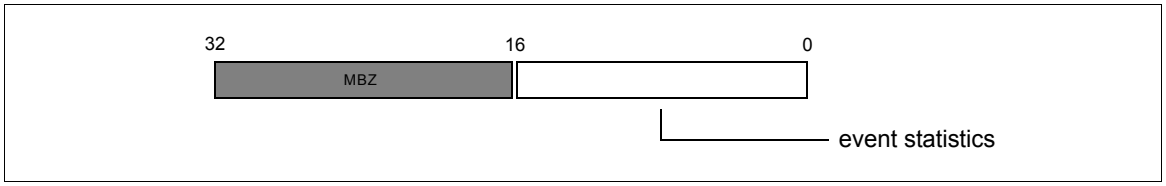

Figure 61 The Event statistics register

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

2.3.7 External condition delay register

This register is used to adjust the relative delay of the GEM's external trigger condition (see Section 2.4.2). The delay has a range of seven-bits. There is a fixed delay of three clock cycles (3) in the arrival of the condition at the GEM until its arrival to the GEM's window-opening logic. Therefore, the delay register has an implicit offset of three clock cycles. This implies a minimum delay of three clock cycles and a maximum delay of 127+3 (decimal) clock cycles (nominally 6.5 microseconds).

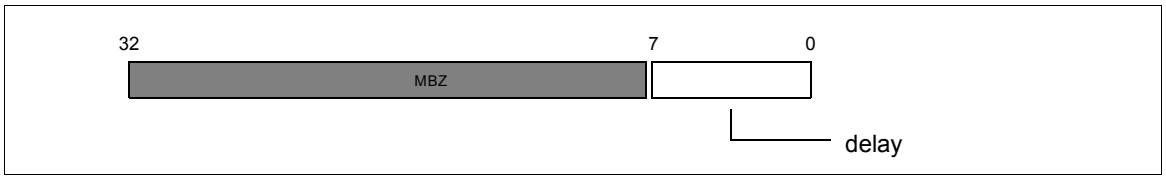

Figure 62 External trigger condition delay register

2.4 Window registers

This section incorporates all the registers whose configuration has a global affect over all the functional blocks of the GEM.

Table 10 The GEM control registers

Name	Address	Access	Description
WINDOW_WIDTH	00	R/W	Masks for window width
WINDOW_OPEN_MASK	01	R/W	Masks for window open sources
Total	**2**		

2.4.1 Window Width

This register determines, once a trigger window is opened, how long it will continue to stay open. The width is specified in units of system clocks, where one system clock is nominally equal to 50 nanoseconds. As the field is five bits wide, the maximum value is 31 SYSCLKs (or 1550 nanoseconds). While permitted by the interface a value of *zero* for this field is not recommended.

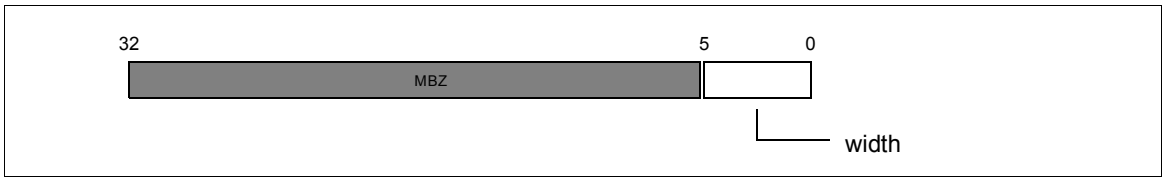

Figure 63 Window Width register

2.4.2 Window Open Mask register

This register determines which input trigger signals are allowed to open a trigger window. (See Section 1.5.) This register is a 8-bit mask, whose bit offsets correspond to the seven different conditions and one ROI signal which are capable of opening a window. The association between bit offset and condition or signal is defined by Figure 64. If the bit at a

particular offset is *set*, the corresponding condition or signal can open the window. If the bit is *clear*, the corresponding condition or signal, even if present, can *not* open the trigger window.

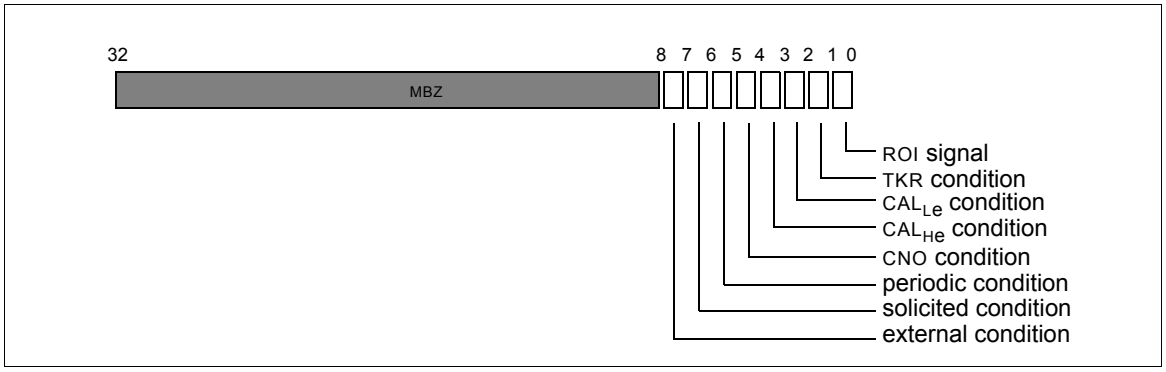

Figure 64 Window Open Mask register

2.5 The TAM Generator registers

The message encoded and transmitted by the GEM when it declares a trigger is called the *Trigger Accept Message* (TAM). The structure and content of this message is described in Section 1.8.6. User control of message content is managed through two entities:

- The sequence register, which is described in Section 2.3.4.
- The *sixteen Message Engines* of the TAM generator, which are discussed here.

These sixteen engines constitute a “palette” of message generators available to the GEM each time it wishes to declare a trigger. The GEM determines which engine to invoke for which trigger by running the *address* of a Message Engine through a lookup table. The index into this table is a *Condition Summary*. This lookup table is contained in the *Scheduler*. In short, each and every window turn will activate one of sixteen Message Engines and the particular engine activated is determined by the Condition Summary associated with that window turn. Once activated, a Message Engine actually satisfies two purposes:

- Prescales the window turn associated with the message engine
- Provides the ***Context*** value for the message (if generated)

Each engine has an associated register, used to configure the engine. Each register, independent of engine, is identical in structure and length (32 bits). Thus, there are sixteen registers to configure sixteen engines. Each one of these registers is called an *Message template*. Their enumeration and addresses are given in Table 11:

Table 11 The TAM generator registers

Name	Address	Access	Description
ENGINE-0	00	R/W	Message template for engine $_0$
ENGINE-1	01	R/W	Message template for engine$_1$
ENGINE-2	02	R/W	Message template for engine$_2$
ENGINE-3	03	R/W	Message template for engine$_3$
ENGINE-4	04	R/W	Message template for engine$_4$
ENGINE-5	05	R/W	Message template for engine$_5$
ENGINE-6	06	R/W	Message template for engine$_6$
ENGINE-7	07	R/W	Message template for engine$_7$
ENGINE-8	08	R/W	Message template for engine$_8$
ENGINE-9	09	R/W	Message template for engine$_9$
ENGINE-A	0A	R/W	Message template for engine$_{10}$
ENGINE-B	0B	R/W	Message template for engine$_{11}$
ENGINE-C	0C	R/W	Message template for engine$_{12}$
ENGINE-D	0D	R/W	Message template for engine$_{13}$
ENGINE-E	0E	R/W	Message template for engine$_{14}$
ENGINE-F	0F	R/W	Message template for engine$_{15}$
Total	**16**		

The structure of any one register is illustrated in Figure 65:

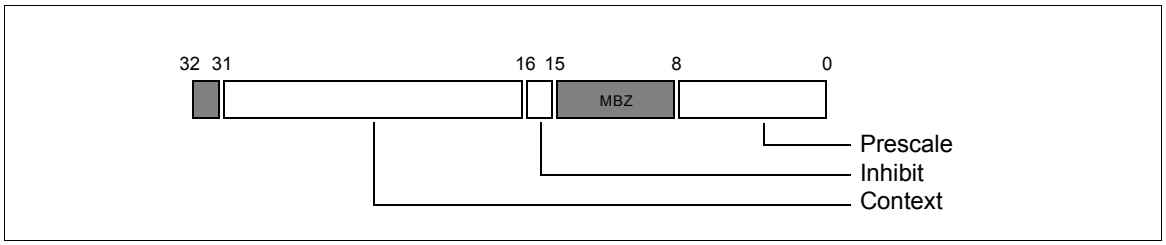

Figure 65 Trigger Accept Message Engine template register

Prescale: Each message engine contains an 8-bit prescaler, implemented as a countdown register. The message engine will determine whether the window turn will cause a trigger by *prescaling* the window turn associated with the message engine. Only when the prescaler expires (goes to *zero*) will a trigger be generated.[1] The resulting trigger rate is equal to $1/(n + 1)$ of the window rate into the engine,

1. Subject to *busy* conditions.

where *n* corresponds to the value of this field. For example, a value of 0 leaves the output rate unchanged from the input rate, and a value of 255 (decimal) prescales the input rate by roughly 0.4%. The value of this field is applied to the prescaler under three conditions:

— A module *reset*

— Updating (writing) the field

— When an engine is activated and its prescale is expired

Inhibit: Determines whether any window turn which activates the engine will generate a trigger, *independent* of its prescale value. If this field is *clear*, the engine's ability to generate a trigger is determined by its prescaler. If this field is *set*, window turns which activate this engine will *not* generate a trigger, independent of the state of the prescaler.

Context: The context value for any TAM generated through this engine. A description of the both the structure and the usage of its fields is found in Section 1.8.6.1. The ***Tag*** field is *read-only* and has a value of *zero* (0). The high order field of the destination address is *read-only* and is always *set* (1).

2.6 Trigger statistics

The GEM provides four counters to tally and record its own performance and two counters which keep track of the singles rates of the tile's of the ACD. The setup and current value of each one of these counters is available through the different registers of the statistics block These registers are enumerated in both Table 12. On overflow, these counters *wrap* (begin re-counting from zero).

Table 12 The Trigger Statistics block registers

Name	Address	Description
LIVETIME	00	1/deadtime
PRESCALED	01	Window turns discarded due to *prescaling*
DISCARDED	02	Window turns discarded due to *busy*
SENT	03	Window turns which result in a sent message
TILE_COUNTERS	04	Actual value of both counters
TILE_0	05	Specifies the tile number to be associated with the first counter
TILE_1	06	Specifies the tile number to be associated with the second counter
CNO_COUNTERS	07	Actual value of both counters
CNO_0	08	Specifies the FREE board number associated with the first counter
Total	**13**	

Table 12 The Trigger Statistics block registers

Name	Address	Description
CNO_1	09	Specifies the FREE board number associated with the second counter
1-PPS	10	Contains the 1-PPS counter and its last sampled time
TIMEBASE	11	Contains the current value of the GEM's internal time-base
DEAD_ZONED	12	The number of times a condition was asserted in the dead-zone
Total	**13**	

2.6.1 Performance counters

The GEM provides *five* correlated counters to tally and record its own performance. The interface to each of these counters is available through five of the registers of the trigger statistics block (see Table 12):

i. LIVETIME at address *zero* (0)
ii. PRESCALED at address *one* (1)
iii. DISCARDED at address *two* (2)
iv. SENT at address *three* (3)
v. DEAD_ZONED at address *twelve* (12)

Because these counters are correlated the behaviour of their corresponding registers when either read or written is somewhat different then other registers of the GEM. When any *one* of these five registers is *written*, the value to be written is ignored and the corresponding five counters are reset to *zero*. Note, that a write will not affect the value of these registers. When the LIVETIME register is *read*, the counters maintaining all five statistics are sampled and held in their respective registers. Consequently, the performance counters will continue to increment independent of how many times the corresponding registers are read and the register values will not change until the LIVETIME register is re-read.

These five counters may be broken into two classes: one which deals with the LAT's *livetime*, and four which deal with *window* statistics.

2.6.1.1 Livetime

The modules of the LAT which participate in event data taking[1] are as follows:

- The sixteen towers (through their TEM's)
- The ACD (through its AEM)
- The GEM itself (in transmission of both TAM and event)

1. Are nodes on the Event fabric.

Depending on a module's resource capability, it may or may not be capable of event readout when the GEM finds a triggerable condition in the detector. On these occasions a module is said to be *busy*. A module communicates its busy state to the GEM using signals on dedicated wires. The total time spent by the LAT busy is called its *deadtime*. The GEM maintains track of the LAT's deadtime and both samples and reports its value whenever an event is declared (See Section 4.10.). However (paradoxically), an event cannot be declared if the LAT is busy. Therefore, rather then maintaining deadtime, the GEM maintains the LAT's the total time spent by the LAT *not* busy. This metric is called the LAT's *livetime*. Livetime is tallied by 25-bit counter which increments in units of system clock ticks (50 nanoseconds) and whose current value is returned in Figure 66:

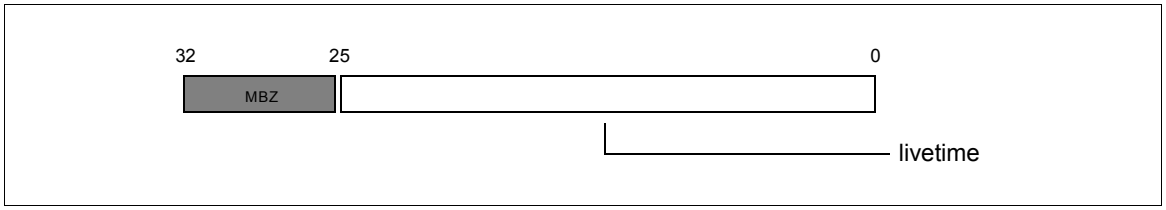

Figure 66 Livetime statistics register

2.6.1.2 Dead-Zoned

It takes a minimum of two clock cycles for the GEM to recover from forming one window before it can potentially form another window. If a condition occurs within this interval, it is ignored. This interval is called the "dead zone". The GEM maintains a counter of the number of times that a window failed to open because one or more *enabled* conditions occurred within the dead zone. If a condition occurs within the dead-zone, but is *not* enabled, the counter will *not* increment. If more then one enabled condition occurs simultaneously within the dead-zone, the counter increments by only *one*. The current value of the counter is returned in the register illustrated in Figure 67.

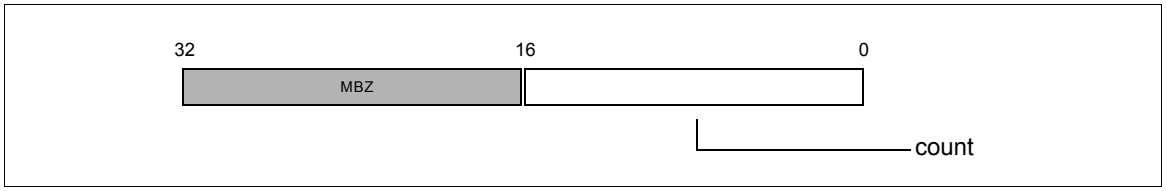

Figure 67 Dead-Zoned register

2.6.1.3 Window turns

Both the towers and ACD send Trigger Inputs to the GEM. The presence of any one of these inputs potentially causes the GEM to open and close its trigger window. An opening and closing of a trigger window is called a *window turn*. (See sections 1.5 and 1.8.2.) Each window turn of the GEM will be deposed of in one of three fashions. It will be:

prescaled: The window turn does *not* cause a trigger to be generated as the prescaler of the engine corresponding to the Condition Summary is not expired. (See Section 2.4.) The prescaled rate is tallied by a *24*-bit counter.

discarded: The engine prescaler corresponding to the Condition Summary expired; however the window turn does *not* cause a trigger to be generated as at least one node on the event fabric was asserting "busy." The busy rate is tallied by a *24*-bit counter.

sent: The engine prescaler corresponding to the Condition Summary *is* expired, "busy" is *not* asserted, and consequently a TAM is broadcast to all the nodes of the trigger fabric. The sent rate is tallied by a *16*-bit counter.

The GEM counts separately each of these occurrences. Their sum equals the total number of window turns. The last sampled value of each counter is contained in the registers illustrated below:

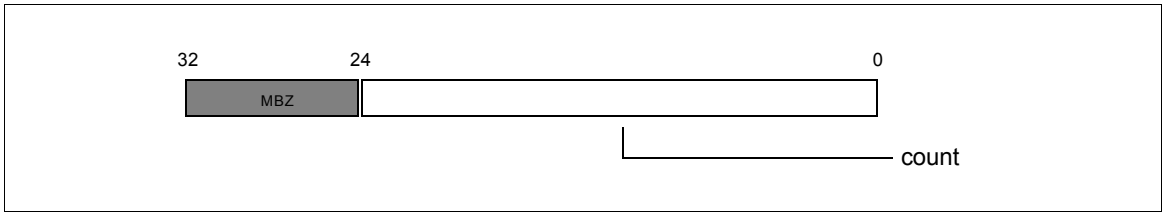

Figure 68 Prescaled statistics register

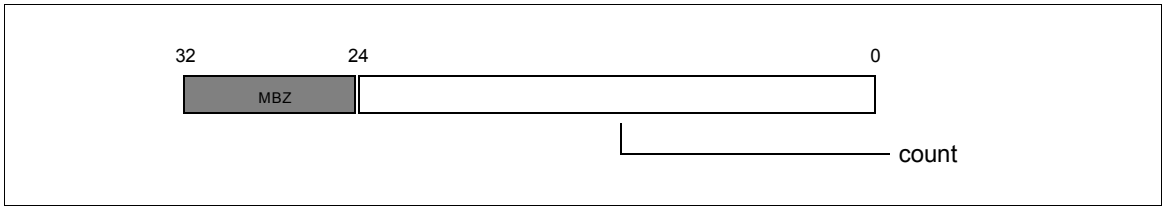

Figure 69 Discarded statistics register

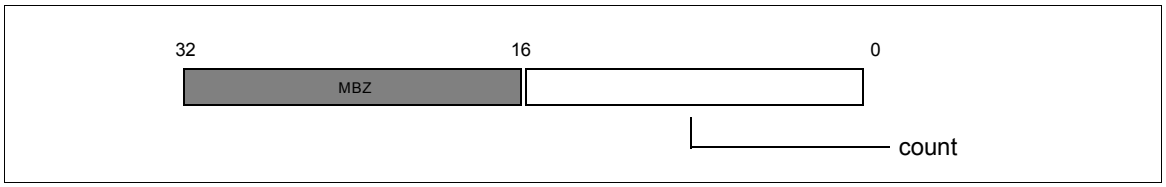

Figure 70 Sent statistics register

2.6.2 Tile counters

The GEM provides two 16-bit counters to tally and record the number of times a signal is present in any *two* of the ACD's 108 tiles. The setup, control, and monitoring for these counters is accomplished through three registers of the statistics block as enumerated in Table 12. One register (Figure 71) maintains the current values of the two counters and the other two registers (Figure 72 and Figure 73) are used to define which two of the 108 tiles should be counted. The value written to any of these two registers determines the actual tile to be counted. This value is expressed as a tile *number*, which varies from `00` to `6B` (hexadecimal). If an invalid tile number is specified, the access is ignored. The relationship between tile number

and tile name is enumerated in Table A.1. As it is necessary to correlate the two counters, write access to these registers is somewhat special:

- When *either* of the two tile registers are written, both counters are reset (*zeroed*).
- When the counter register is written, the value to be written is ignored and both counters are reset (*zeroed*).

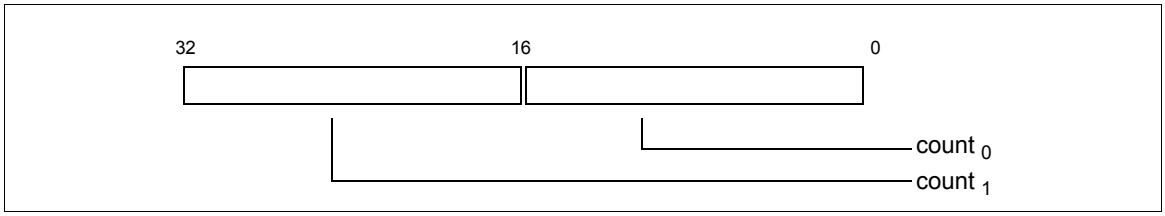

Figure 71 Tile counters

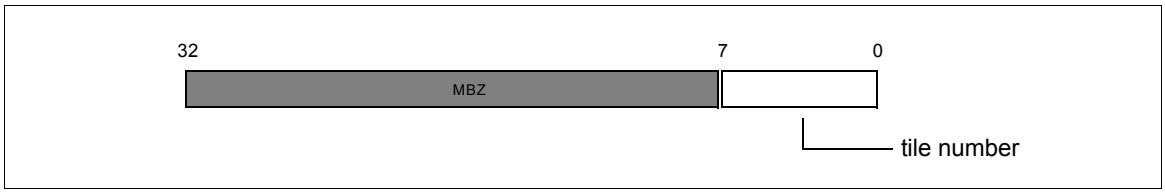

Figure 72 Tile to be counted by first tile counter

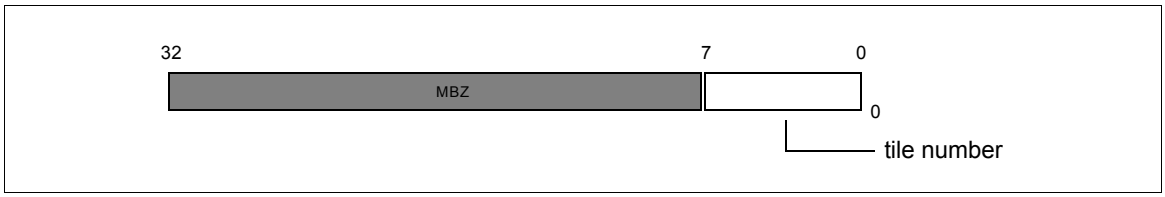

Figure 73 Tile to be counted by second tile counter

2.6.3 CNO counters

The GEM provides two 16-bit counters to tally and record the number of times a CNO signal is present in any *two* of the ACD's twelve different FREE boards. The setup, control, and monitoring for these counters is accomplished through three registers of the statistics block as enumerated in Table 12. One register (Figure 74) maintains the current values of the two counters and the other two registers (Figure 75 and Figure 76) are used to define which two of the CNO signals from the twelve different FREE boards should be counted. The value written to any of these two registers determines the actual CNO signal counted. This value is expressed as a free board *number*, which varies from `00` to `0C` (hexadecimal). If an invalid free board number is specified, the access is ignored. The relationship between FREE board number and FREE board name is enumerated in Table A.1. As it is necessary to correlate the two counters, write access to these registers is somewhat special:

- When *either* of the two tile registers are written, both counters are reset (*zeroed*).

- When the counter register is written, the value to be written is ignored and both counters are reset (*zeroed*).

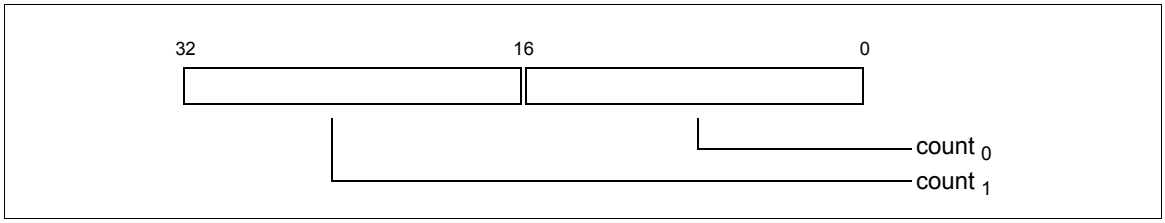

Figure 74 CNO counters

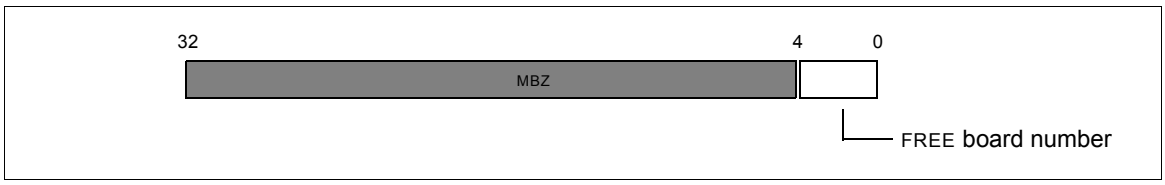

Figure 75 CNO signal to be counted by first CNO counter register

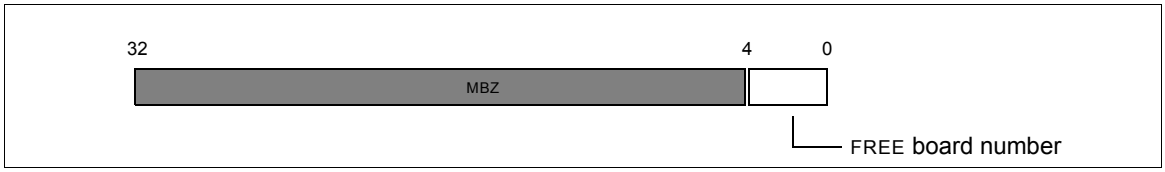

Figure 76 Tile to be counted by second tile counter register

2.6.4 1-PPS timers

The GEM maintains both a counter and a register which are updated upon the arrival of the spacecraft's 1-PPS signal:

seconds: This field counts the number of arrived 1-PPS signals since the GEM was reset; thus, it increments each time a 1-PPS signal arrives. As the 1-PPS signal arrives once per second, this register is nominally a count of the number of seconds since the GEM was reset. This register is 7 bits wide. Note that this counter simply continues on overflow.

1-PPS time: Each time the 1-PPS signal arrives, the GEM's timebase (discussed in Section 1.9) is sampled and saved in this register. Thus, this register contains the time, in system clock ticks (50 nano-seconds), of the last arrived 1-PPS signal. As the timebase is a 25-bit counter, this field is 25 bits wide.

The structure of the register which contains both of these quantities is illustrated in Figure 77. Note: this register is *Read-Only*.

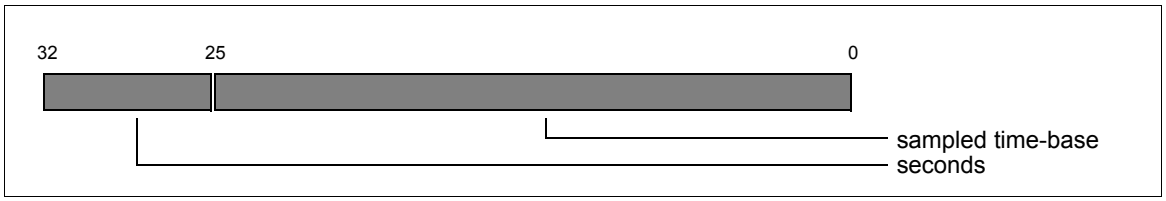

Figure 77 1-PPS timing register

2.6.5 Time-Base

The GEM maintains its own time-base as a free-running, 25-bit counter, incrementing at the system clock rate (nominally 20 MHZ). Thus, one count in the time-base corresponds to 50 nanoseconds. The counter simply continues on overflow. The current value of the time-base is returned in the register illustrated in Figure 78. Note: this register is *Read-Only*.

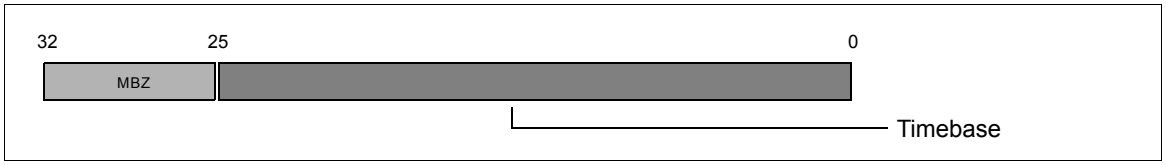

Figure 78 Time-Base register

2.7 Scheduler registers

The Scheduler contains the lookup table which is used to map a Condition Summary to a Message Engine. This process is described in detail within sections 1.7.3 and 2.4. However, in short, each time the Scheduler performs a window turn, it generates a Condition Summary. The Scheduler treats the summary as a 8-bit number used to *index* into the lookup table. The *entries* of this table correspond to the addresses of one of the sixteen engines on the TAM generator. The user configures the entries of the lookup table through the thirty-two registers of the Scheduler. These registers are enumerated in Table 13:

Table 13 The Scheduler registers

Name	Address	Description
CONDITIONS_00-07	00	Engine lookup table entries
CONDITIONS_08-0F	01	Engine lookup table entries
CONDITIONS_10-17	02	Engine lookup table entries
CONDITIONS_18-1F	03	Engine lookup table entries
CONDITIONS_20-27	04	Engine lookup table entries
CONDITIONS_28-2F	05	Engine lookup table entries
CONDITIONS_30-37	06	Engine lookup table entries
CONDITIONS_38-3F	07	Engine lookup table entries
CONDITIONS_40-47	08	Engine lookup table entries
CONDITIONS_48-4F	09	Engine lookup table entries
CONDITIONS_50-57	10	Engine lookup table entries
CONDITIONS_58-5F	11	Engine lookup table entries
CONDITIONS_60-67	12	Engine lookup table entries
CONDITIONS_68-6F	13	Engine lookup table entries
CONDITIONS_70-77	14	Engine lookup table entries
CONDITIONS_78-7F	15	Engine lookup table entries
CONDITIONS_80-87	16	Engine lookup table entries
CONDITIONS_88-8F	17	Engine lookup table entries
CONDITIONS_90-97	18	Engine lookup table entries
CONDITIONS_98-9F	19	Engine lookup table entries
CONDITIONS_A0-A7	20	Engine lookup table entries
CONDITIONS_A8-AF	21	Engine lookup table entries
CONDITIONS_B0-B7	22	Engine lookup table entries
CONDITIONS_B8-BF	23	Engine lookup table entries
CONDITIONS_C0-C7	24	Engine lookup table entries
CONDITIONS_C8-CF	25	Engine lookup table entries
CONDITIONS_D0-D7	26	Engine lookup table entries
CONDITIONS_D8-DF	27	Engine lookup table entries
Total	**32**	

Table 13 The Scheduler registers

Name	Address	Description
CONDITIONS_E0-E7	28	Engine lookup table entries
CONDITIONS_E8-EF	29	Engine lookup table entries
CONDITIONS_F0-F7	30	Engine lookup table entries
CONDITIONS_F8-FF	31	Engine lookup table entries
Total	**32**	

As each register is 32 bits wide and it requires 4 bits to specify an engine address, each register is responsible for *eight* entries of the lookup table. The structure of any one of the sixteen registers is illustrated in Figure 79. For example, assume the user wishes to use the Condition Summary described in Section 1.7.3 (index 0A) to activate Engine number five (5). Then the value 5 would be written into register CONDITIONS_00-0F starting at offset 40 (decimal). *Note: This example is not right and needs to be corrected.*

The mapping between register and Condition Summary is as follows:

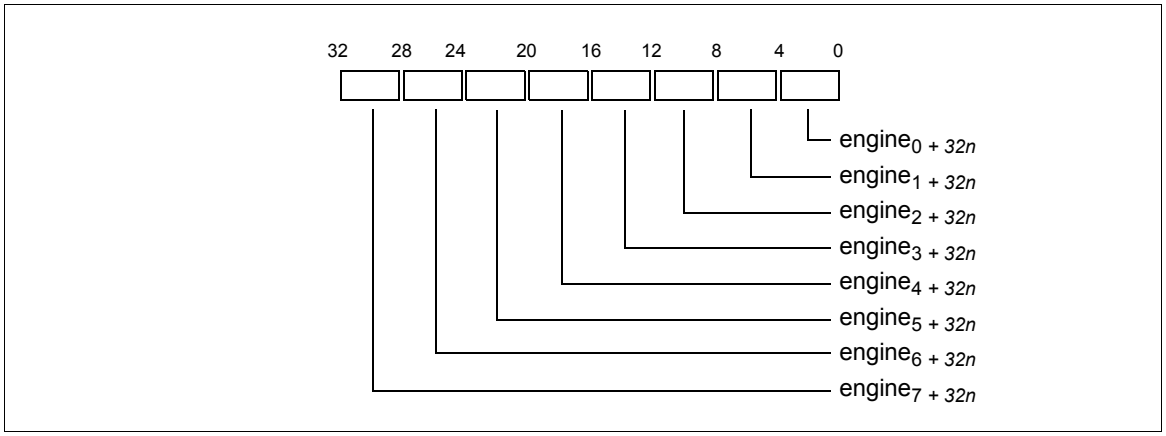

Figure 79 Lookup table (Scheduler register)

2.8 Region-Of-Interest (ROI) Generator registers

The Region-Of-Interest (ROI) generator registers are used to define the association between each of the 108 tile pairs[1] of the ACD and up to sixteen Regions-Of-Interest. The definition of a region depends on whether the input signals from the ACD are used by the GEM as a veto or as a trigger. (See Figure 53 and Section 1.3.1.) In any case, any one register of the ROI generator contains the region definitions for two tiles. (See Figure 80.) Each tile requires a single bit to

1. Here, the expression “tile pair” and tile are considered synonyms.

specify its participation in the region, which implies any one tile requires a 16 bit field to specify all sixteen regions.

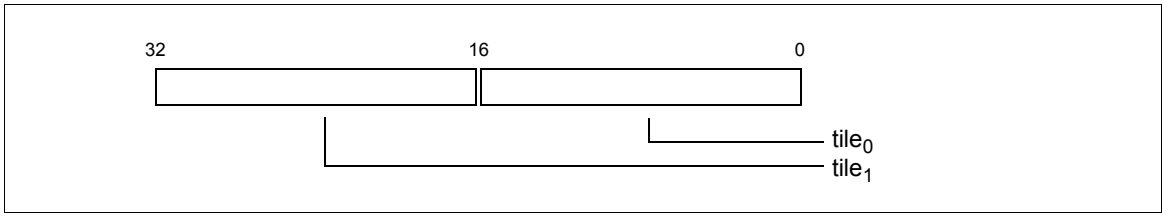

Figure 80 ROI generator register

At two tiles per register, the ROI generator requires 54 registers to specify all the tiles. The relationship between tile and register is enumerated in Table 14:

Table 14 The ROI Generator registers

	defines tiles:			defines tiles:			defines tiles:	
Address	**0**	**1**	**Address**	**0**	**1**	**Address**	**0**	**1**
00	000	001	01	002	003	02	004	010
03	011	012	04	013	014	05	020	021
06	022	023	07	024	030	08	031	032
09	033	034	10	040	041	11	042	043
12	044	NA2	13	NA3	100	14	101	102
15	103	104	16	110	111	17	112	113
18	114	120	19	121	122	20	123	124
21	130	NA4	22	NA5	200	23	201	202
24	203	204	25	210	211	26	212	213
27	214	220	28	221	222	29	223	224
30	230	NA6	31	NA7	300	32	301	302
33	303	304	34	310	311	35	312	313
36	314	320	37	321	322	38	323	324
39	330	NA8	40	NA9	400	41	401	402
42	403	404	43	410	411	44	412	413
45	414	420	46	421	422	47	423	424
48	430	NA0	49	NA1	500	50	501	502
51	503	600	52	601	602	53	603	NA10

2.8.1 The region used as a tower shadow

If the ACD is used as a veto, each region corresponds to one of the LAT's 16 towers and any one region is used to define which tiles shadow a particular tower. As the LAT has sixteen towers, a 16-bit mask for each tile is used to establish this association. This mask and the relationship between tower number and bit offset is illustrated in Figure 81:

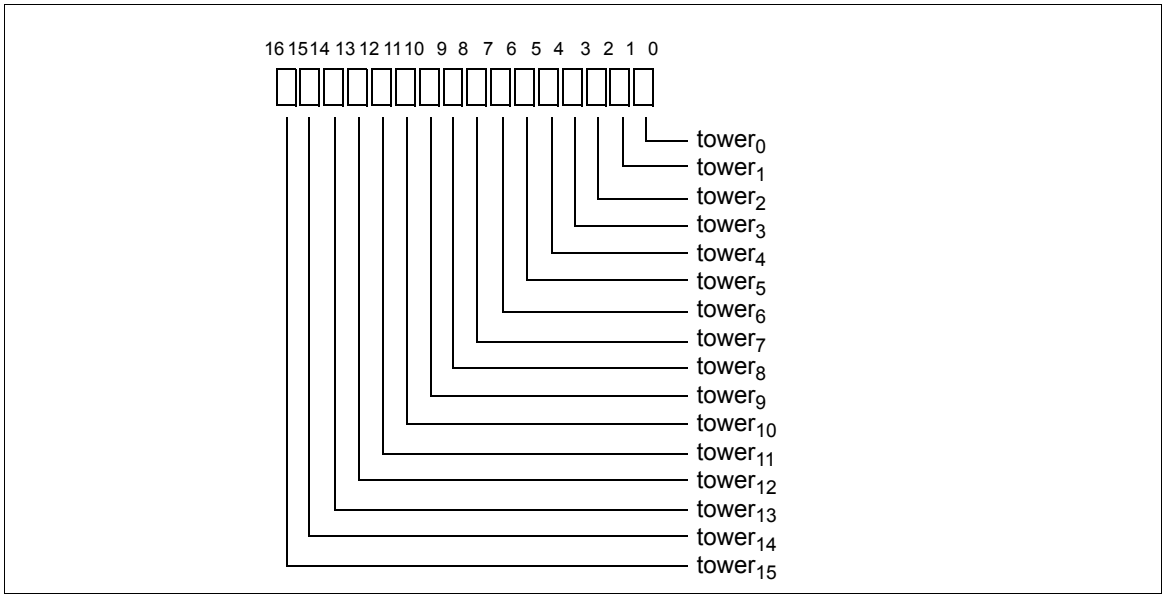

Figure 81 Veto and Tower Association mask

Each bit offset corresponds to whether its corresponding tower is shadowed by a tile. If a bit at a particular offset is *set*, the tower is shadowed by the tile.

2.8.2 The region used to form coincidences

If the ACD is used as a trigger, the sixteen regions are divided into eight region pairs. For each region pair, one region is defined as *odd* and the other as *even*. Odd and even region pairs are used to form tile coincidences. Coincidences between one or more pairs can then be used to trigger a readout of the LAT. A two-bit field is necessary to specify a tile's association with any one particular region pair as illustrated in Figure 82:

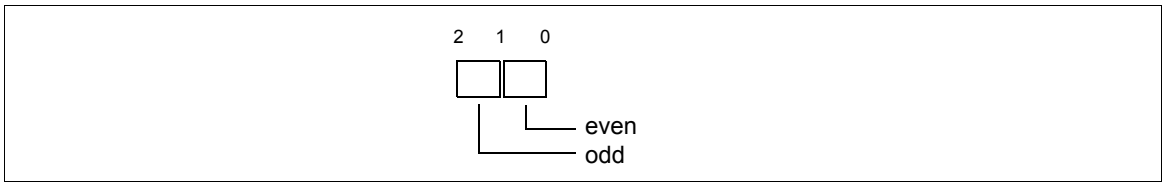

Figure 82 Coincidence pair

If a bit at a specified offset is *set*, the tile participates in the specified region. If both fields are *clear*, the tile will not be associated with any region. Note that a tile can participate in *both* odd

and even regions. (This feature is useful in using a single tile as a trigger.) Finally, as there are eight possible region pairs, the relationship between region pair and bit offset for any one tile is illustrated in Figure 83:

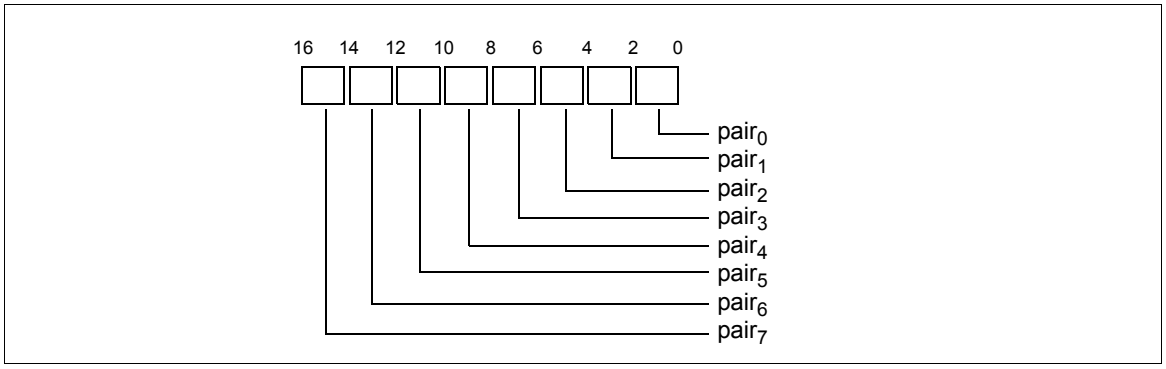

Figure 83 Veto and coincidence pair association

2.9 Input enable registers

These registers allow individual masking of the set of trigger input signals which are presented to the GEM. Table 15 enumerates the registers and their addresses:

Table 15 Input enable registers

Name	Address	Description
TOWERS_0-3	00	Four Tower inputs
TOWERS_4-7	01	Four Tower inputs
TOWERS_8-11	02	Four Tower inputs
TOWERS_12-15	03	Four Tower inputs
ACD_CNO	04	FREE board CNO inputs
TILES_000-013	05	Nine tile enables
TILES_014-032	06	Nine tile enables
TILES_033-NA3	07	Nine tile enables
TILES_100-113	08	Nine tile enables
TILES_114-NA5	09	Nine tile enables
TILES_200-213	10	Nine tile enables
TILES_214-NA7	11	Nine tile enables
TILES_300-313	12	Nine tile enables
Total	**19**	

Table 15 Input enable registers

Name	Address	Description
TILE_314-NA9	13	Nine tile enables
TILES_400-413	14	Nine tile enables
TILES_414-NA1	15	Nine tile enables
TILES_500-NA10	16	Nine tile enables
TOWER_BUSY	17	"Busy" signals from the sixteen towers
EXTERNAL	18	External trigger condition enable
Total	**19**	

2.9.1 Tower enable registers

These registers allow masking of the trigger input signals for the sixteen towers of the LAT. Each of the sixteen towers (as serviced through its TEM) produces three trigger input signals. Thus, there are a total of 48[1] input signals from 16 towers. Each tower's three trigger input signals are organized as shown in Figure 84.

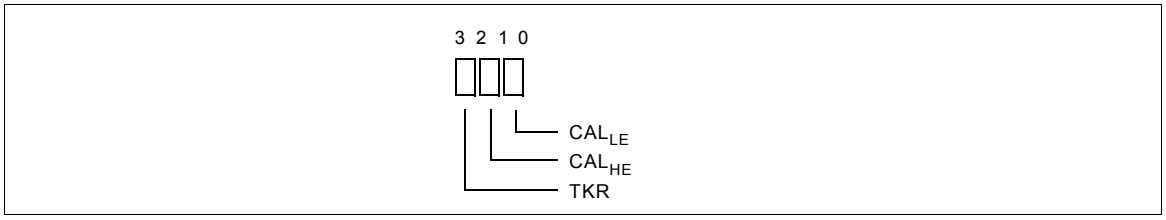

Figure 84 One tower's trigger input masking

Any one of the four tower enable registers specifies the input enables for four of the sixteen towers as illustrated in Figure 85. A bit offset corresponds to a particular signal's mask. If the bit at a specified offset is *set*, the corresponding signal is *enabled*. If the bit is *clear*, the signal is *disabled*.

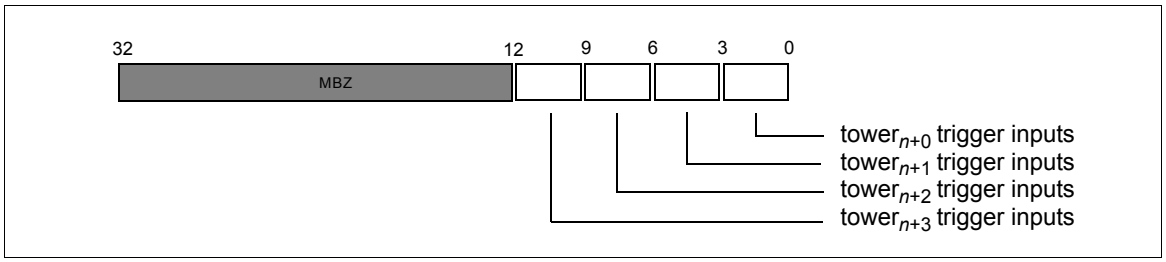

Figure 85 Towers enable register

1. There are also the *busy* signals which are masked using the register described in Figure 88.

2.9.2 ACD CNO enable register

This register allows independent masking of the CNO signals from each of the twelve FREE boards of the ACD. A bit offset corresponds to this signal's mask from a particular FREE board. If the bit at a specified offset is *set*, the CNO signal for that board is *enabled*. If the bit is *clear*, the signal is *disabled*. The format of this register is illustrated in Figure 86.

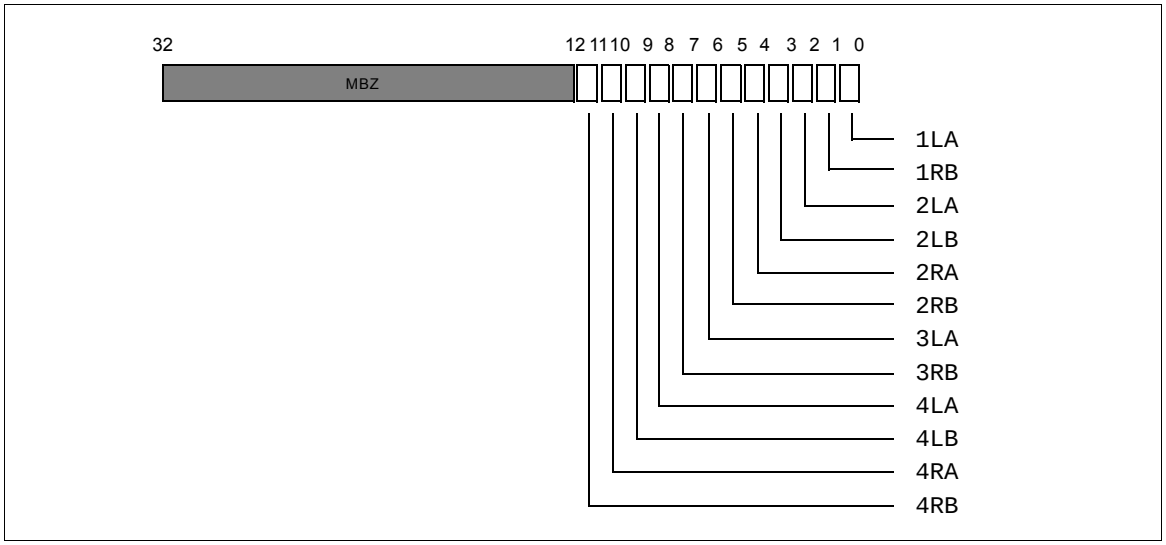

Figure 86 ACD CNO enable register

2.9.3 Tile enable registers

Each side of each of the 108[1] tile pairs constituting the ACD's signals are independently maskable. There are twelve registers to specify the masking for the entire set of 108 tiles. Each register defines the enables for *nine* tiles: nine one-bit fields for the "A" side of the tiles and the other nine one-bit fields for the "B" side of the tiles. This structure is illustrated by Figure 87. If the bit at a specified offset is *clear*, the signal corresponding to that side is masked *off*. If the bit is *set*, the signal corresponding to that side is *enabled*.

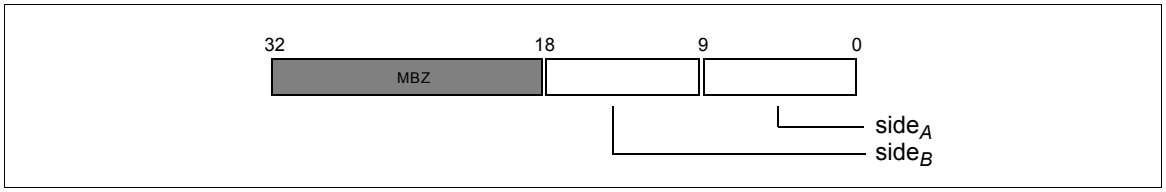

Figure 87 Structure of a tile enable register

1. Actually, there are 97 tiles, but here, I also include the NAs.

The registers are arranged in increasing tile number. The mapping between a register and its bit offsets and the corresponding FREE board and channel are enumerated in the twelve tables below:

Table 16 Enables register for tiles 000 through 013

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
000	2LA	6	00	2LB	11	09
001	2LA	12	01	2LB	5	0A
002	2LA	17	02	2LB	0	0B
003	2RA	6	03	2RB	11	0C
004	2RA	11	04	2RB	6	0D
010	2LA	7	05	2LB	10	0E
011	2LA	13	06	2LB	4	0F
012	2RA	4	07	2RB	13	10
013	2RA	5	08	2RB	12	11

Table 17 Enables register for tiles 014 through 032

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
014	2RA	10	00	2RB	7	09
020	2LA	8	01	2LB	9	0A
021	2LA	14	02	2LB	3	0B
022	2LA	15	03	2LB	2	0C
023	4LA	15	04	4LB	2	0D
024	4LA	9	05	4LB	8	0E
030	4RA	10	06	4RB	7	0F
031	4RA	5	07	4RB	12	10
032	4RA	4	08	4RB	13	11

Table 18 Enables register for tiles 33 through NA3

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
033	4LA	14	00	4LB	3	09
034	4LA	8	01	4LB	9	0A
040	4RA	11	02	4RB	6	0B
041	4RA	6	03	4RB	11	0C
042	4LA	17	04	4LB	0	0D
043	4LA	13	05	4LB	4	0E
044	4LA	7	06	4LB	10	0F
NA2	4RA	13	07	4RB	16	10
NA3	4RA	16	08	2LB	16	11

Table 19 Enables register for tiles 100 through 113

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
100	2LA	1	00	1RB	3	09
101	1LA	6	01	1RB	7	0A
102	1LA	9	02	1RB	8	0B
103	1LA	10	03	1RB	11	0C
104	1LA	14	04	4RB	1	0D
110	2LA	0	05	1RB	2	0E
111	1LA	5	06	1RB	6	0F
112	1LA	8	07	1RB	9	10
113	1LA	11	08	1RB	12	11

Table 20 Enables register for tiles 114through NA5

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
114	1LA	15	00	4RB	0	09
120	1LA	0	01	1RB	1	0A
121	1LA	4	02	1RB	5	0B
122	1LA	7	03	1RB	10	0C
123	1LA	12	04	1RB	13	0D
124	1LA	16	05	1RB	17	0E
130	1LA	17	06	1RB	0	0F
NA4	1LA	1	07	1RB	16	10
NA5	1LA	3	08	1RB	14	11

Table 21 Enables register for tiles 200 through 213

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
200	2LA	5	00	2LB	12	09
201	2LA	11	01	2LB	6	0A
202	2RA	3	02	2RB	14	0B
203	2RA	7	03	2RB	10	0C
204	2RA	12	04	2RB	5	0D
210	2LA	3	05	2LB	14	0E
211	2LA	10	06	2LB	7	0F
212	2RA	2	07	2RB	15	10
213	2RA	8	08	2RB	9	11

Table 22 Enables register for tiles 214 through NA7

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
214	2RA	14	00	2RB	3	09
220	2LA	2	01	2LB	15	0A
221	2LA	9	02	2LB	8	0B
222	2RA	0	03	2RB	17	0C
223	2RA	9	04	2RB	8	0D
224	2RA	15	05	2RB	2	0E
230	2RA	17	06	2LB	17	0F
NA6	2LA	16	07	2LB	13	10
NA7	2RA	13	08	2RB	16	11

Table 23 Enables register for tiles 300 through 313

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
300	3LA	14	00	2RB	1	09
301	3LA	10	01	3RB	11	0A
302	3LA	9	02	3RB	8	0B
303	3LA	6	03	3RB	7	0C
304	4LA	1	04	3RB	3	0D
310	3LA	15	05	2RB	0	0E
311	3LA	11	06	3RB	12	0F
312	3LA	8	07	3RB	9	10
313	3LA	5	08	3RB	6	11

Table 24 Enables register for tiles 314 through NA9

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
314	4LA	0	00	3RB	2	09
320	3LA	16	01	3RB	17	0A
321	3LA	12	02	3RB	13	0B
322	3LA	7	03	3RB	10	0C
323	3LA	4	04	3RB	5	0D
324	3LA	0	05	3RB	1	0E
330	3LA	17	06	3RB	0	0F
NA8	3LA	3	07	3RB	14	10
NA9	3LA	1	08	3RB	16	11

Table 25 Enables register for tiles 400 through 413

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
400	4RA	12	00	4RB	5	09
401	4RA	7	01	4RB	10	0A
402	4RA	3	02	4RB	14	0B
403	4LA	12	03	4LB	5	0C
404	4LA	6	04	4LB	11	0D
410	4RA	14	05	4RB	3	0E
411	4RA	8	06	4RB	9	0F
412	4RA	2	07	4RB	15	10
413	4LA	11	08	4LB	6	11

Table 26 Enables register for tiles 414 through NA1

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
414	4LA	5	00	4LB	12	09
420	4RA	15	01	4RB	2	0A
421	4RA	9	02	4RB	8	0B
422	4RA	0	03	4RB	17	0C
423	4LA	10	04	4LB	7	0D
424	4LA	3	05	4LB	14	0E
430	4RA	17	06	4LB	17	0F
NA0	4LA	2	07	4LB	15	10
NA1	4LA	16	08	4LB	16	11

Table 27 Enables register for tiles 500 through NA10

Tile	PMT "A"			PMT "B"		
	board	chnl	offset	board	chnl	offset
500	3LA	13	00	1RB	4	09
501	3LA	2	01	1RB	15	0A
502	1LA	2	02	3RB	15	0B
503	1LA	13	03	3RB	4	0C
600	2LA	4	04	4RB	4	0D
601	4RA	1	05	2LB	1	0E
602	2RA	1	06	4LB	1	0F
603	4LA	4	07	2RB	4	10
NA10	2RA	16	08	4LB	13	11

2.9.4 Tower busy enable register

These registers allow masking of the *busy* signals for the sixteen towers of the LAT. Each of the sixteen towers (as serviced through its TEM), produces one busy signal. The structure of the register to mask busy signals is shown in Figure 88.

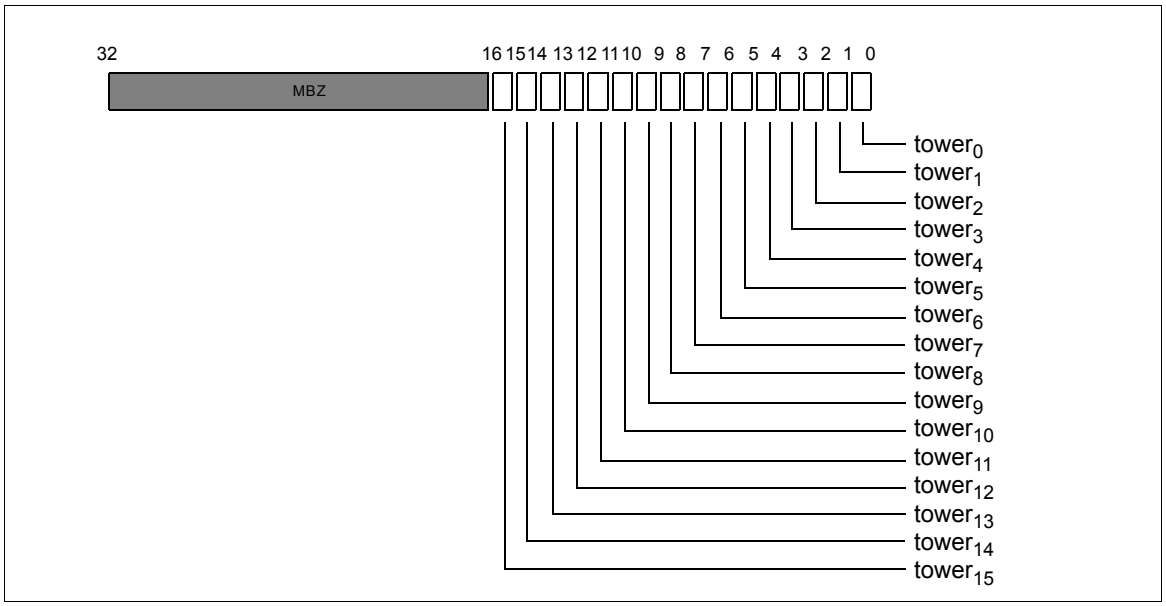

Figure 88 Tower busy enable register

A bit offset corresponds to the mask for its corresponding tower. If the bit at a specified offset is *set*, the corresponding signal is *enabled*. If the bit is *clear*, the signal is *disabled*.

2.9.5 External condition enable register

This register allows masking of the *external* trigger condition signal. The structure of this register is shown in Figure 89.

Figure 89 External Condition enable register

If the register is *set*, the external trigger condition signal is *enabled*. If the register is *clear*, the signal is *disabled*.

Chapter 3
Commanding

3.1 Overview

This chapter describes the remote protocol necessary to access both the registers[1] and functional blocks of the GEM. It follows the Command/Response Protocol discussed in [1]. The registers of the GEM are organized into a hierarchy of six blocks as illustrated in Figure 90.

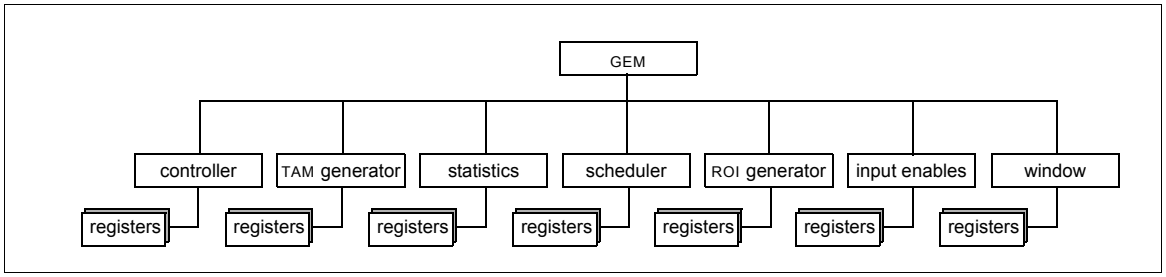

Figure 90 Hierarchy of target types

3.1.1 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 *packet header* described in [1].

1. Enumerated and described in Chapter 2.

3.2 The GEM's access descriptor

Directly following the LATp header of any received command packet is a fixed size, 16-bit structure, called the GEM's ***access descriptor***. This descriptor is a parameterization of the access rules required to address the functional blocks and registers of the GEM. Its structure is illustrated in Figure 91:

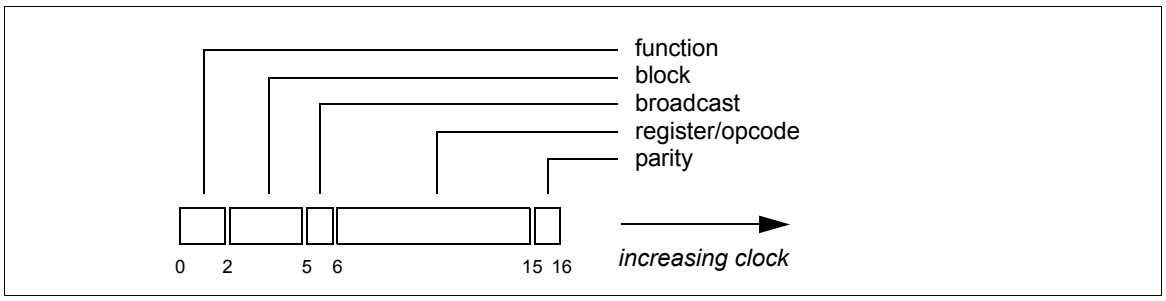

Figure 91 GEM access descriptor

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 enumeration for this field are described in [1].

block: This field enumerates which of the six blocks of the GEM are to be accessed. The correspondence between block type and number is enumerated in Table 28.

broadcast: Determines how the ***register/opcode*** field is interpreted. If this field is *false*, the ***register/opcode*** field is used to determine which register to access. If this field is *true*, the ***register/opcode*** field is ignored and the access descriptor is applied to *all* the registers of the type specified by the ***block*** field. Note: A broadcast operation is *not* permitted if the ***function*** field specifies a `read` operation.

register/opcode: If the ***function*** field has a value of either `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***.

Table 28 Block numbers of the GEM

Name	Number
`controller`	`0`
`TAM generator`	`1`
`statistics`	`2`
`scheduler`	`3`
`ROI Generator`	`4`
`input enables`	`5`
`window`	`6`

3.3 Accessing the controller

An enumeration and description of the registers of the controller block may be found in Section 2.3. All registers of the controller are 32 bits in length.

3.3.1 Dataless commands

Table 29 The controller's dataless commands

Name	Opcode	Description
NOP	0	No operation
RESET	1	Hard reset of the GEM
TRIGGER	2	Solicit trigger
Total	**3**	

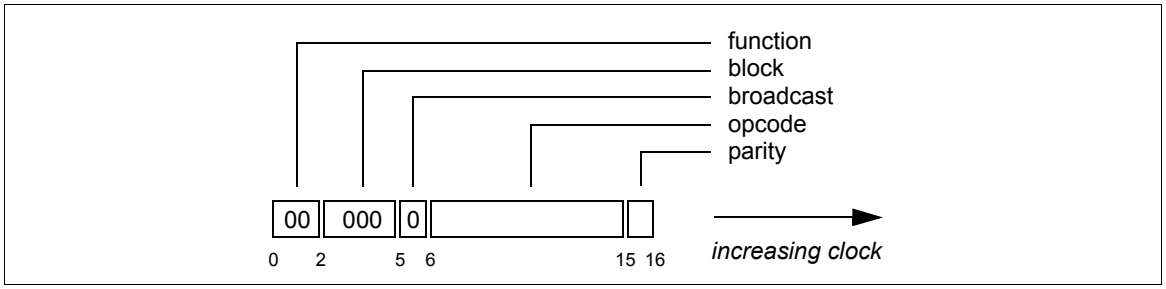

Figure 92 Access descriptor for the controller's dataless commands

`Dataless` functions do *not* require a payload. As a `dataless` function requires no response, the ***respond*** field of the packet is set to *false*.

3.3.2 Load commands

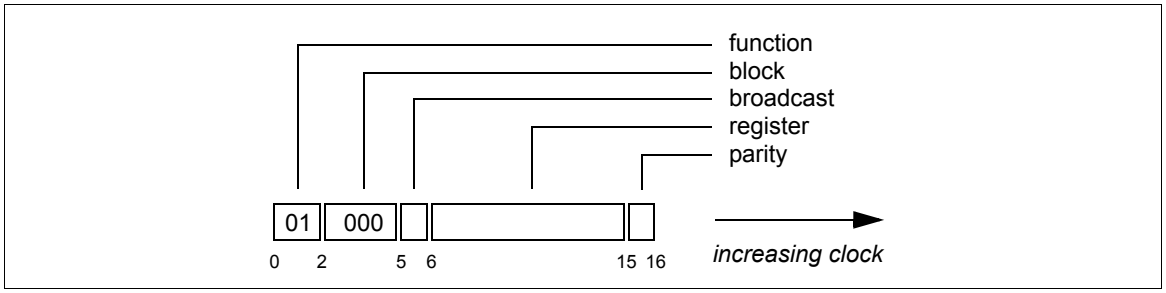

Figure 93 Access descriptor for the controller's register load commands

All registers of the controller are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 94. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

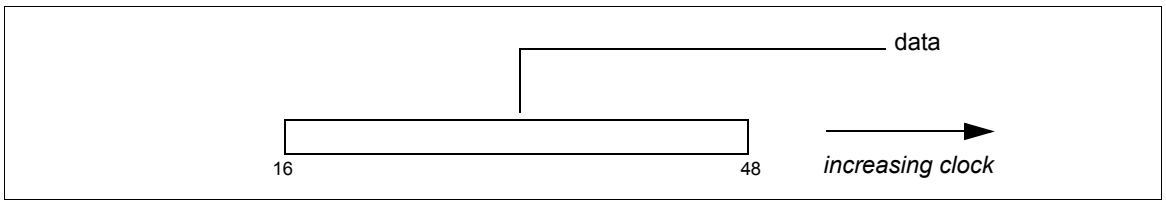

Figure 94 Payload for the controller's register load commands

3.3.3 Read commands

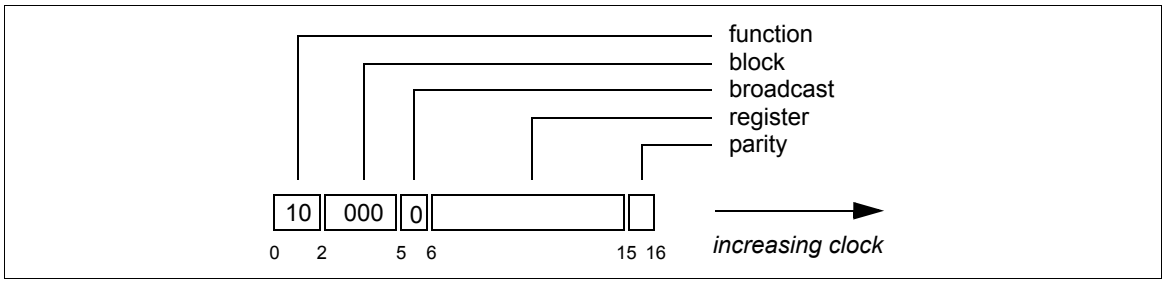

Figure 95 Access descriptor for the controller's register read commands

`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 96.

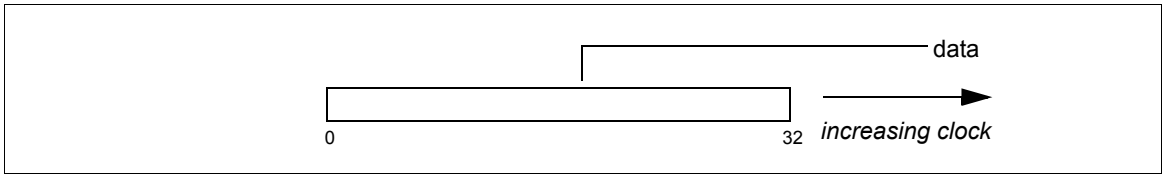

Figure 96 Response to a register read command of the controller

3.4 Accessing the TAM Generator

An enumeration and description of the registers of the TAM generator may be found in Section 2.4. All registers of the TAM Generator are thirty-two bits in length.

3.4.1 Load commands

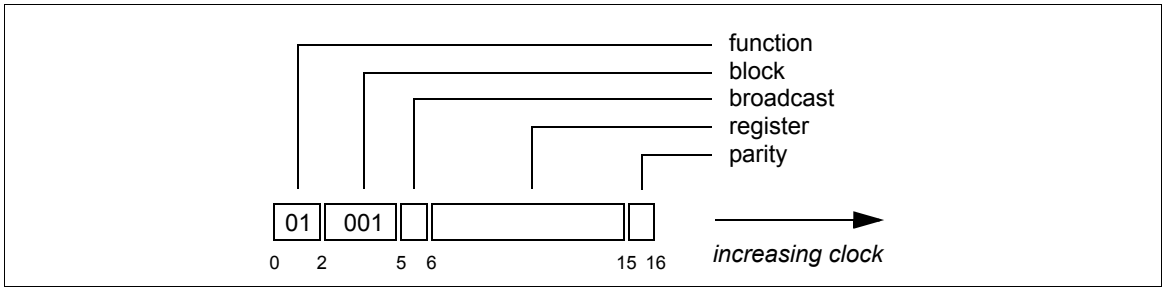

Figure 97 Access descriptor for the TAM Generator's register load commands

All registers of the TAM Generator are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 98. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

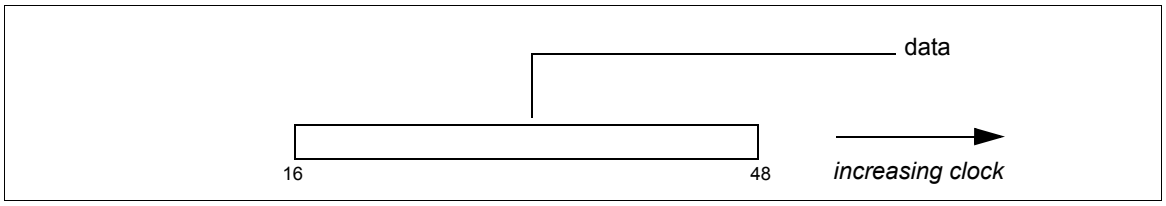

Figure 98 Payload for TAM Generator's register load commands

3.4.2 Read commands

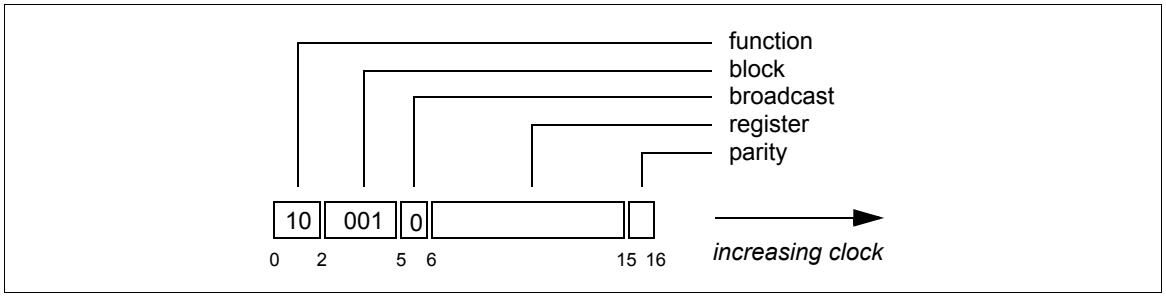

Figure 99 Access descriptor for TAM Generator's register read commands

`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 100.

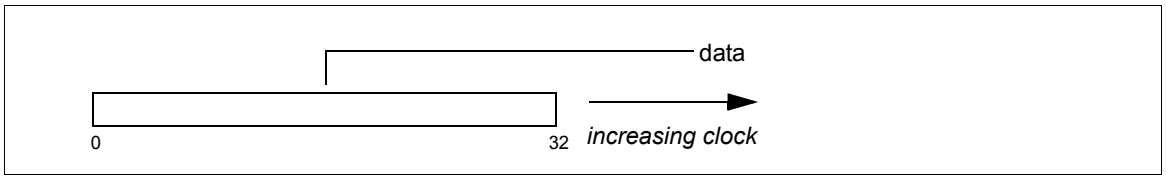

Figure 100 Response to register read commands of the TAM Generator

3.5 Accessing the Trigger Statistics block

An enumeration and description of the registers of the controller block may be found in Section 2.6. All registers of this block are 32 bits in length.

3.5.1 Load commands

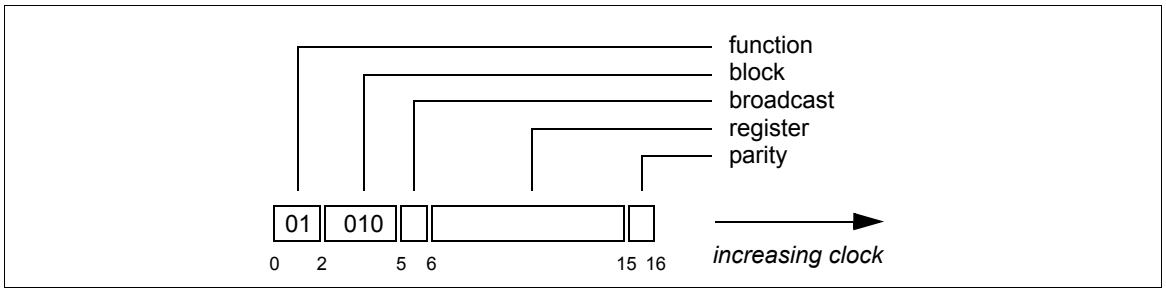

Figure 101 Access descriptor for the statistics block's register load commands

All registers of the trigger block are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 102. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

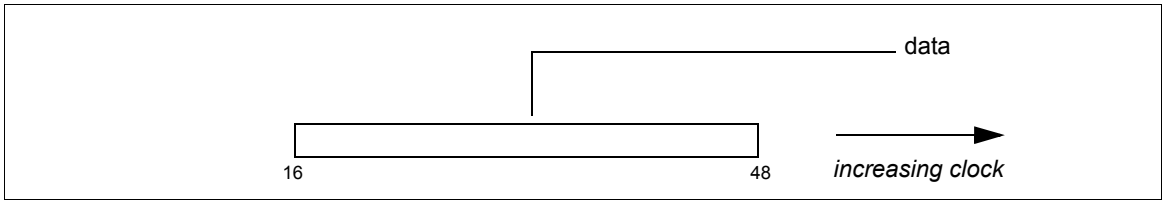

Figure 102 Payload for the statistics block's register load commands

3.5.2 Read commands

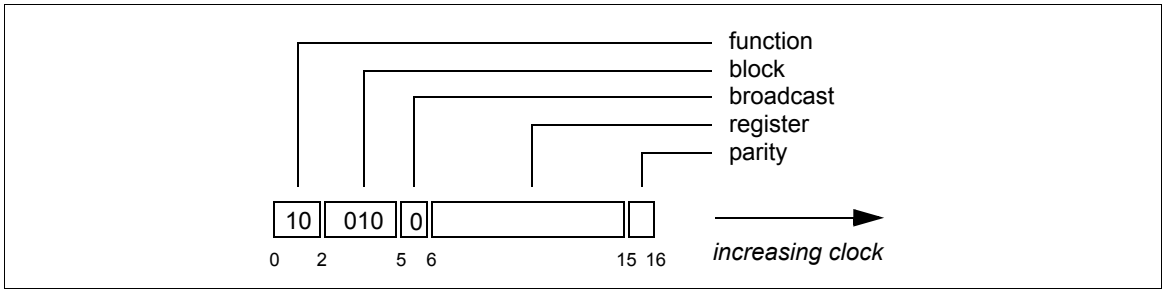

Figure 103 Access descriptor for register read commands of the statistics block

`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 104.

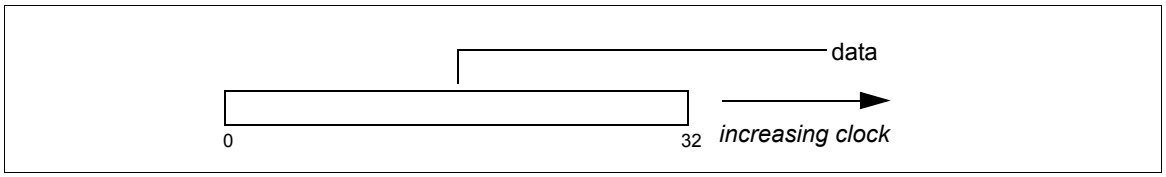

Figure 104 Response to a register read of the statistics block

3.6 Accessing the Scheduler

An enumeration and description of the registers of the Scheduler may be found in Section 2.6. All registers of the Scheduler are 32 bits in length.

3.6.1 Load commands

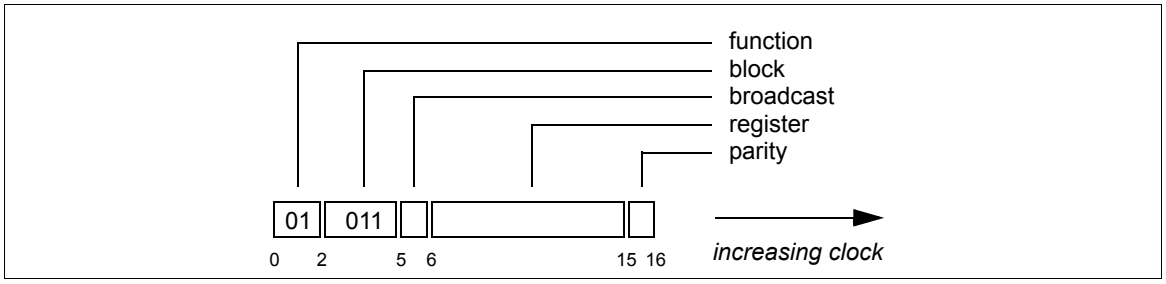

Figure 105 Access descriptor for Scheduler register load commands

All registers of the Scheduler are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 106. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

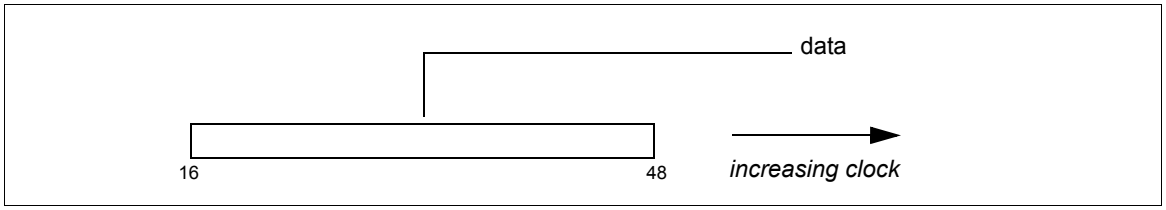

Figure 106 Payload for Scheduler register load commands

3.6.2 Read commands

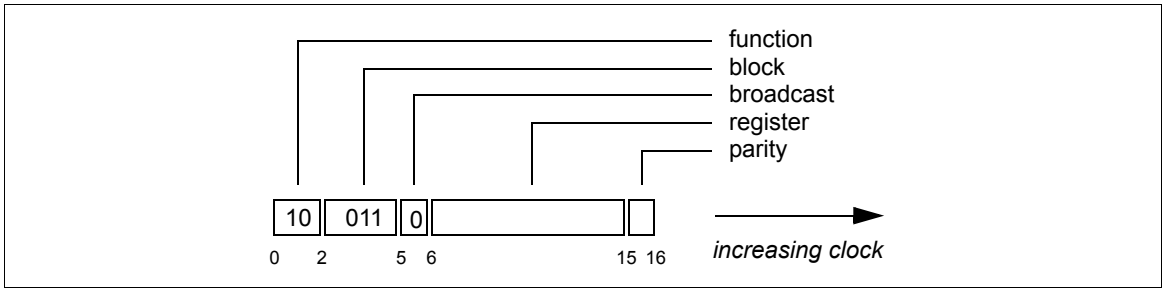

Figure 107 Access descriptor for Scheduler register read commands

`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 108.

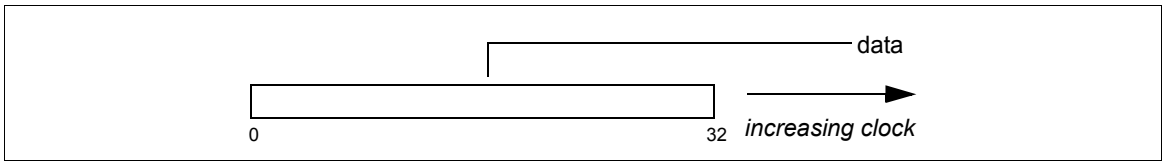

Figure 108 Response to register read commands of the Scheduler

3.7 Accessing the ROI Generator

An enumeration and description of the registers of the ROI Generator may be found in Section 2.8. All registers of the ROI generator are 32 bits in length.

3.7.1 Load commands

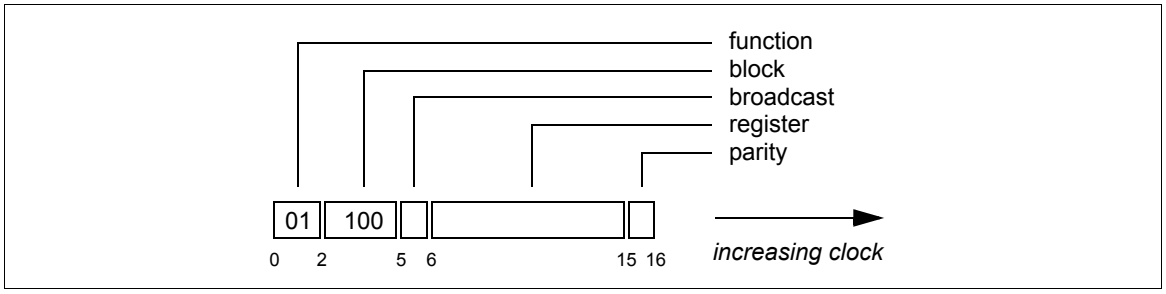

Figure 109 Access descriptor for ROI Generator register load commands

All registers of the ROI generator are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 110. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

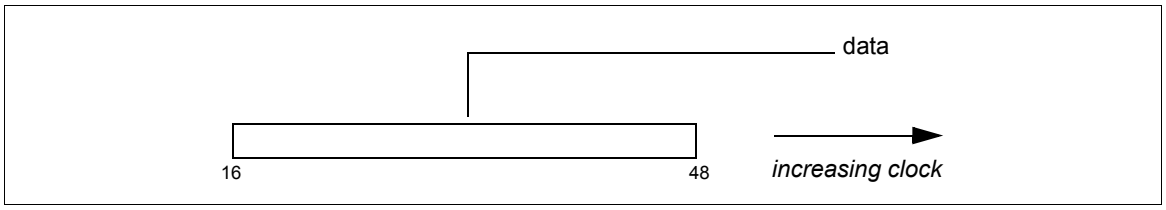

Figure 110 Payload for ROI Generator register load commands

3.7.2 Read commands

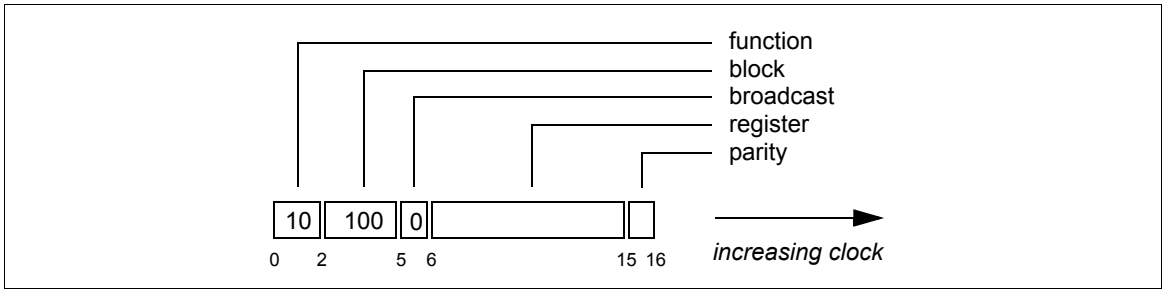

Figure 111 Access descriptor for ROI Generator register read commands

`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 112.

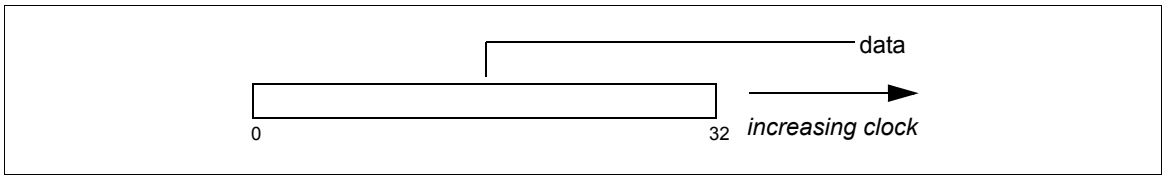

Figure 112 Response to register read commands of the ROI Generator

3.8 Accessing the Input Enables

An enumeration and description of the registers of the Input Enable Registers may be found in Section 2.9. All Input Enable Registers are 32 bits in length.

3.8.1 Load commands

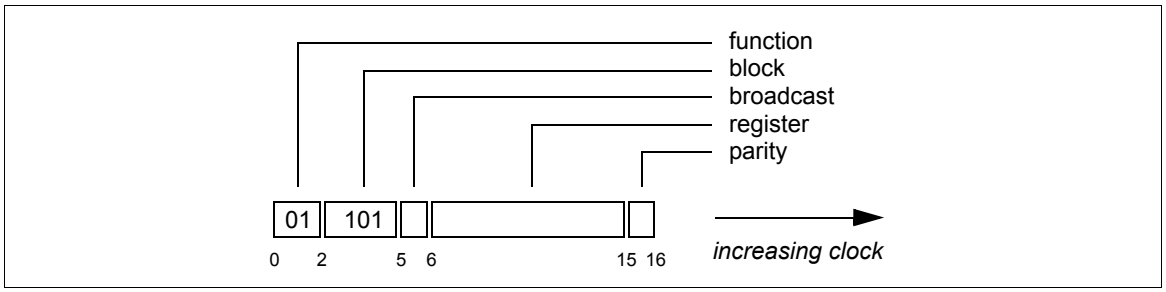

Figure 113 Access descriptor for Input Enable register load commands

All Input Enable Registers are 64 bits long. Consequently, all `Load` functions require a 64-bit payload. The format of this payload is illustrated in Figure 114. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

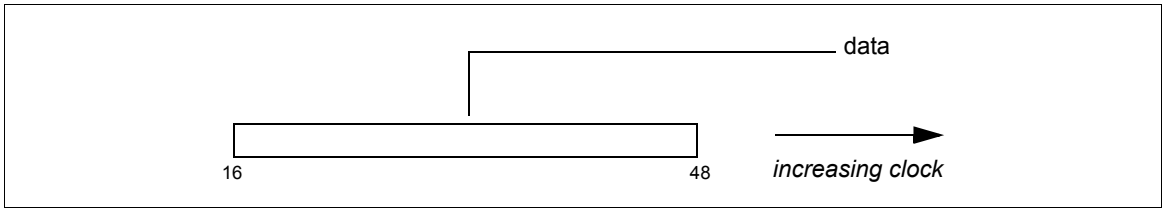

Figure 114 Payload for Input Enable register load commands

3.8.2 Read commands

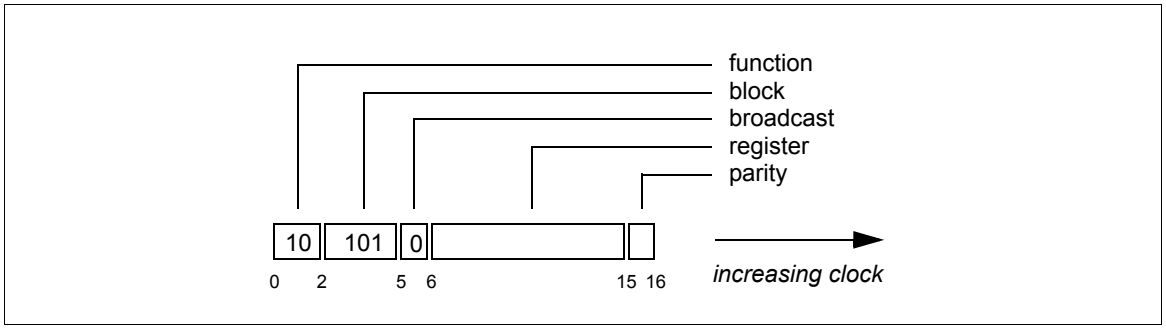

Figure 115 Access descriptor for Input Enable register read commands

`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 116.

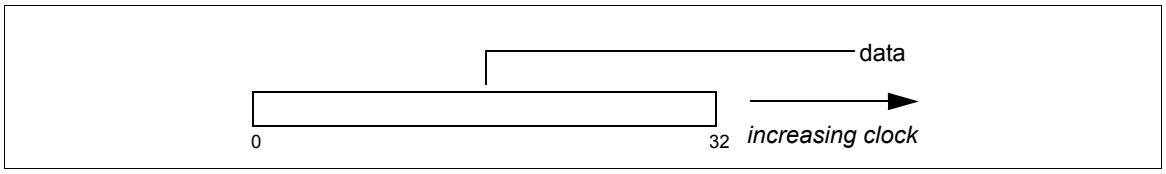

Figure 116 Response to Input Enable register read commands

3.9 Accessing the window block

An enumeration and description of the registers of the window block may be found in Section 2.4. All registers of the window block are 32 bits in length.

3.9.1 Load commands

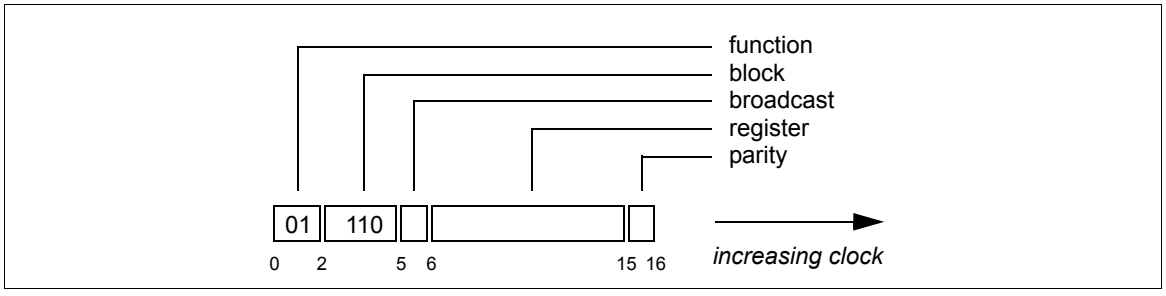

Figure 117 Access descriptor for the window's register load commands

All registers of the window block are 32 bits long. Consequently, all `Load` functions require a 32-bit payload. The format of this payload is illustrated in Figure 118. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

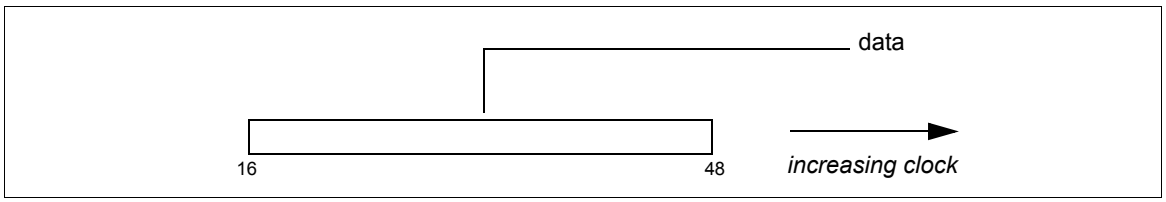

Figure 118 Payload for the window's register load commands

3.9.2 Read commands

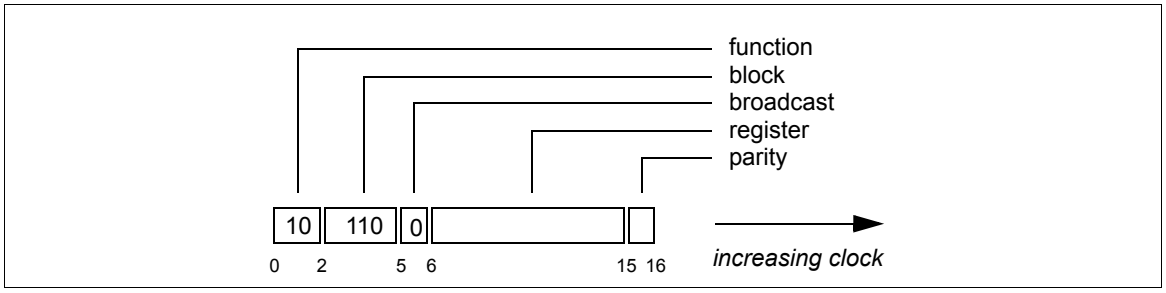

Figure 119 Access descriptor for the window's register read commands

`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 120.

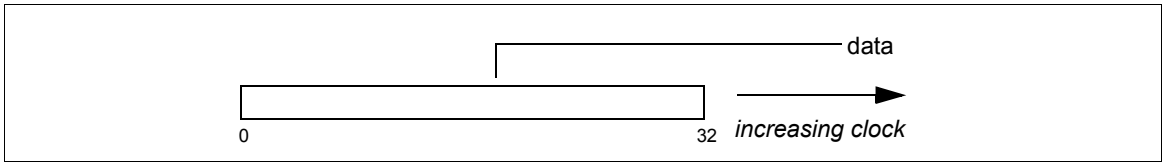

Figure 120 Response to a register read command of the window block

Chapter 4
Events

4.1 The event contribution

This chapter describes the format of the event data generated by the GEM in response to generating a trigger. The *shape* of the GEM's event data is governed by the Event Data Protocol (EDP) described in [1]. The GEM event consists of the required four-byte summary word, followed by a fixed (56 bytes) size block. It does *not* emit either a diagnostic or error contribution; therefore, both the ***diagnostic*** and ***error*** fields of the summary word will always be *clear*. In addition, as the GEM does not transmit a TAM to itself, the ***trigger parity error*** field will also always be *clear*. The overall structure of the GEM event contribution is illustrated in Figure 121:

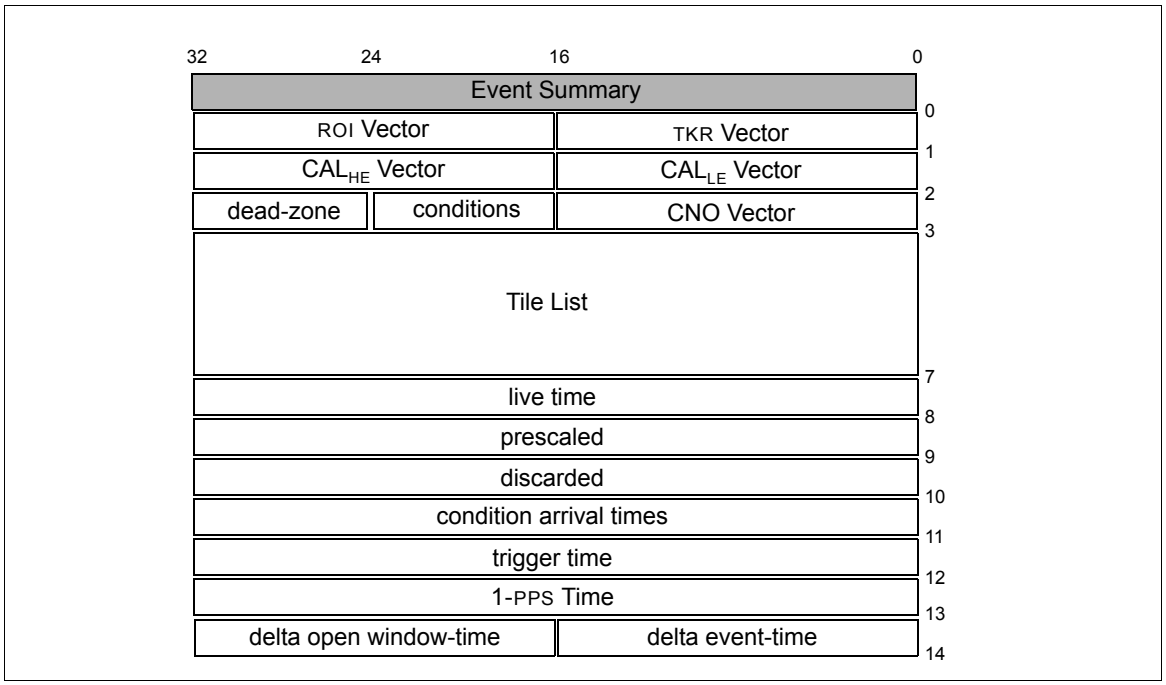

Figure 121 GEM event contribution

4.2 The TKR Vector

The tracker contribution to the event consists of the sampled state of the sixteen (one per tower) tracker signals when the window corresponding to the event was *closed*. This structure is called the TKR Vector and is described in detail in Section 1.6.1.

4.3 The ROI Vector

This contribution to the event consists of the sampled state of the sixteen region signals produced by the ROI Generator when the window corresponding to the event was *closed*. This structure is called the ROI vector and is described in detail in Section 1.6.2. The interpretation of this contribution depends on whether the vector was used as a *veto* or as a *trigger*. If the vector was used as a *veto*, the contribution has a format illustrated by Figure 122:

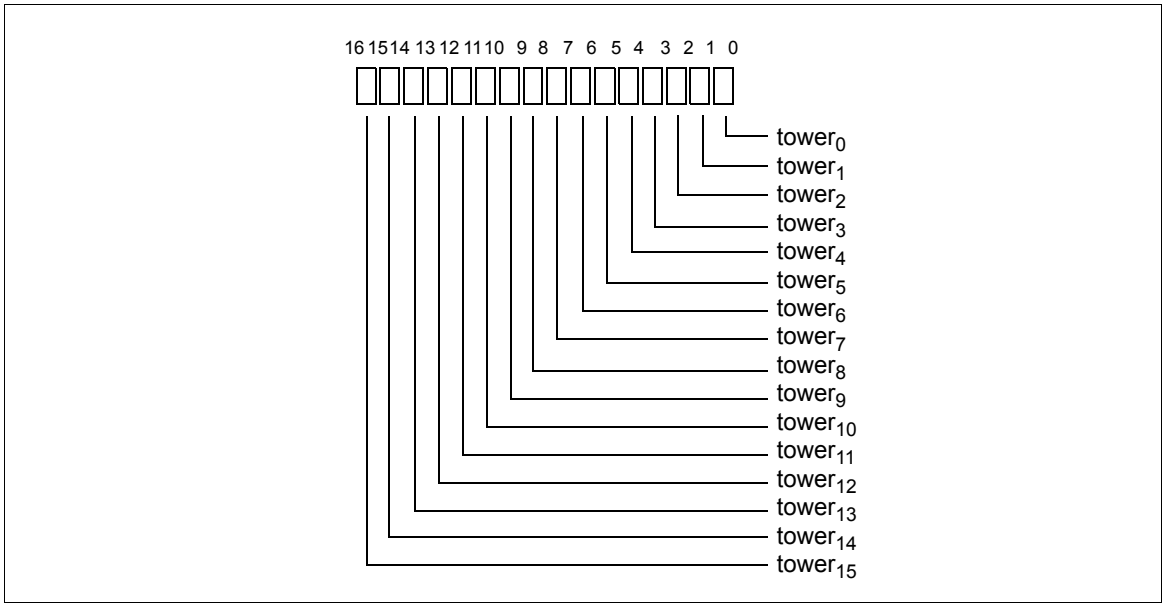

Figure 122 ROI Generator contribution to event (ACD used as veto)

Where each bit offset corresponds to the shadowing of a particular tower by the ACD.

If the ACD was used as a trigger, the contribution has the structure illustrated in Figure 123:

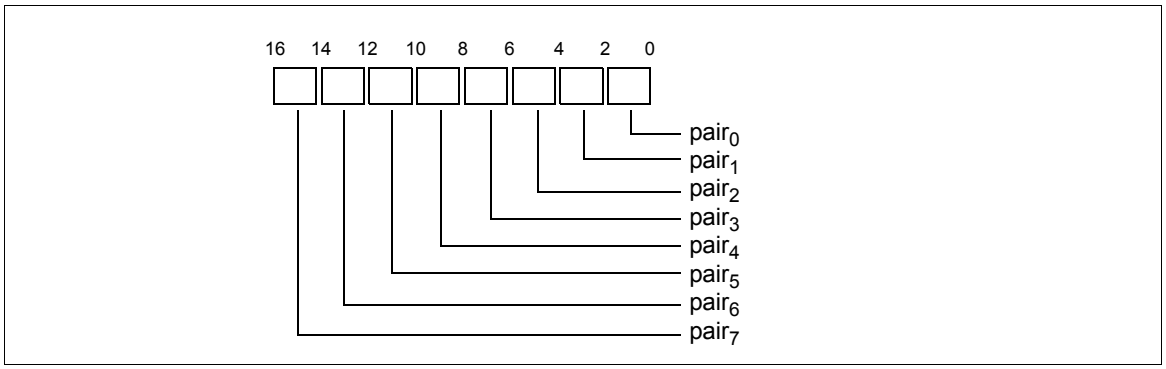

Figure 123 ROI Generator contribution to event (ACD used as trigger)

Where each bit offset corresponds to an odd/even pair coincidence as described in Figure 82.

4.4 The Calorimeter (Low Energy) Vector

This is one of the two contributions of the calorimeter to the event. (See Section 1.6.3.) It consists of the sampled state of the sixteen CAL_{LE} signals (one per tower) when the window corresponding to the event was *closed*.

4.5 The Calorimeter (High Energy) Vector

This is one of the two contributions of the calorimeter to the event. (See Section 1.6.3.) It consists of the sampled state of the sixteen CAL_{HE} signals (one per tower) when the window corresponding to the event was *closed*.

4.6 The CNO Vector

This contribution to the event consists of the sampled state of the twelve CNO signals produced by the ACD (one per FREE board) when the window corresponding to the event was *closed*. This structure is called the CNO vector and is described in detail within Section 1.6.5. The format of the contribution is illustrated in Figure 124:

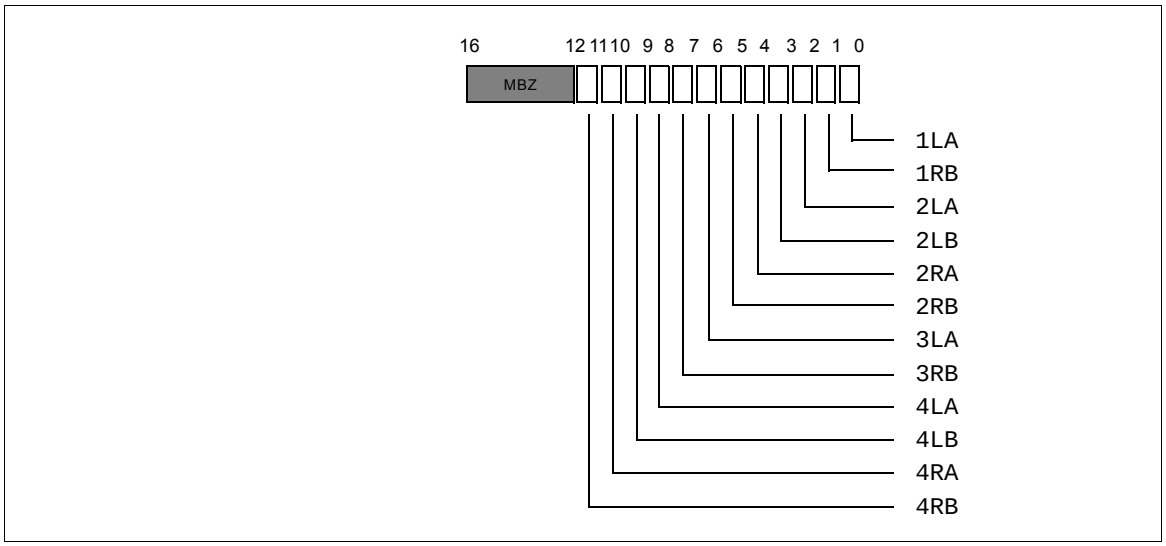

Figure 124 ACD CNO contribution to event

4.7 Condition Summary

This contribution to the event consists of the sampled state of the Condition Summary as determined by the GEM when the window corresponding to the event was *closed*. The summary serves two functions in the generation of an event:

- As a *list* of the reasons why an event was declared.
- As an *index* into the Scheduler lookup table. (See Section 2.4.)

The format of the contribution is illustrated in Figure 125. Each bit offset corresponds to one condition. If a bit at a particular offset is *set*, the corresponding condition was asserted for at least *one* system clock (20 MHZ) within the trigger window. The order of conditions in the summary is specified by Figure 32 within Section 1.7.2.

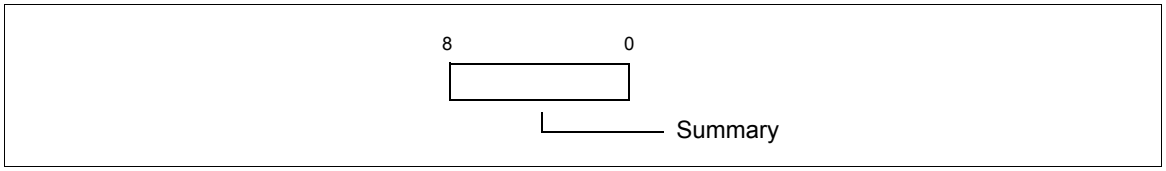

Figure 125 Condition Summary contribution to event

4.8 Dead-Zone count

It takes a minimum of two clock cycles for the GEM to recover from forming one window before it can potentially form another window. If a condition occurs within this interval, it is

ignored. This interval is called the “dead zone”. The GEM maintains a counter of the number of times a condition occurred within the dead zone (see Section 2.6.1.2). At event hold time this counter is sampled and its *least significant 8 bits* are recorded in the contribution illustrated in Figure 126.

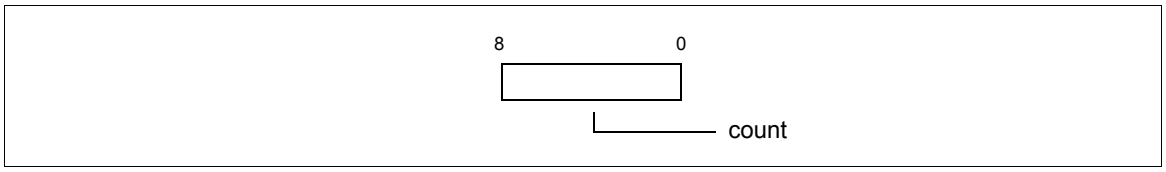

Figure 126 Dead-zone counter contribution to event

4.9 The Tile List

When the window corresponding to the event was *closed*, the GEM samples the state of each of the 97 tiles of the ACD and produces the contribution illustrated in Figure 127. It is convenient for the trigger[1] to map the tile geometry of the ACD into the following six groups:

XZM: Corresponds to the tiles located on the *side* of the ACD labelled -Y in Figure A.1 of Appendix A.

XZP: Corresponds to the tiles located on the *side* of the ACD labelled +Y in Figure A.1 of Appendix A.

YZM: Corresponds to the tiles located on the *side* of the ACD labelled -X in Figure A.1 of Appendix A.

YZP: Corresponds to the tiles located on the *side* of the ACD labelled +X in Figure A.1 of Appendix A.

XY: Corresponds to the tiles located on the *top* of the ACD shown as surrounded by the other four sides in Figure A.1 of Appendix A.

RBN: Corresponds to the *ribbon* tiles of the ACD. The ribbons are represented as the *dark green* channels of the FREE boards of in Figure A.1 of Appendix A.

The tiles corresponding to channels which are not assigned (NA) trail the six groups.

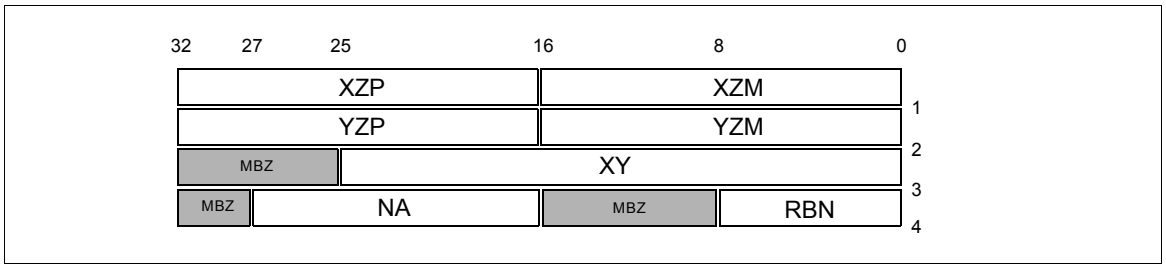

Figure 127 The Veto List contribution to event

1. To increase the efficiency of the Software Filter.

Each bit offset corresponds to the signal for a corresponding tile. If a bit is *set*, the corresponding tile generated a signal for at least *one* system clock (20 MHZ) within the trigger window. The tile assignments for each of the six groups are illustrated in the following sections.

4.9.1 Correspondence between group and tile

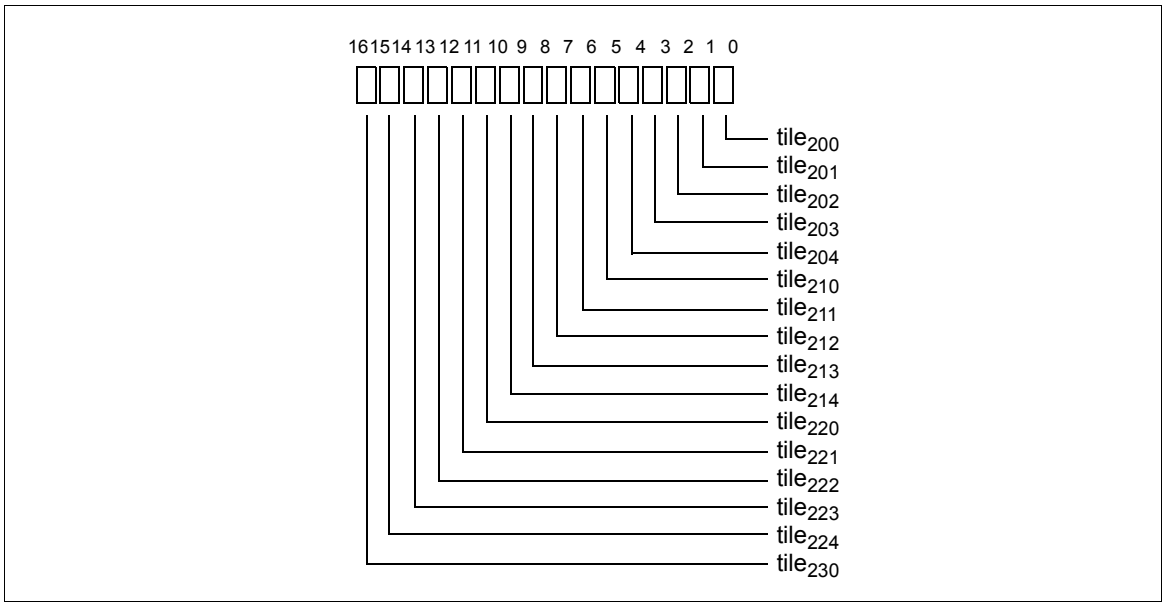

Figure 128 Contribution to event of XZM group in Veto List

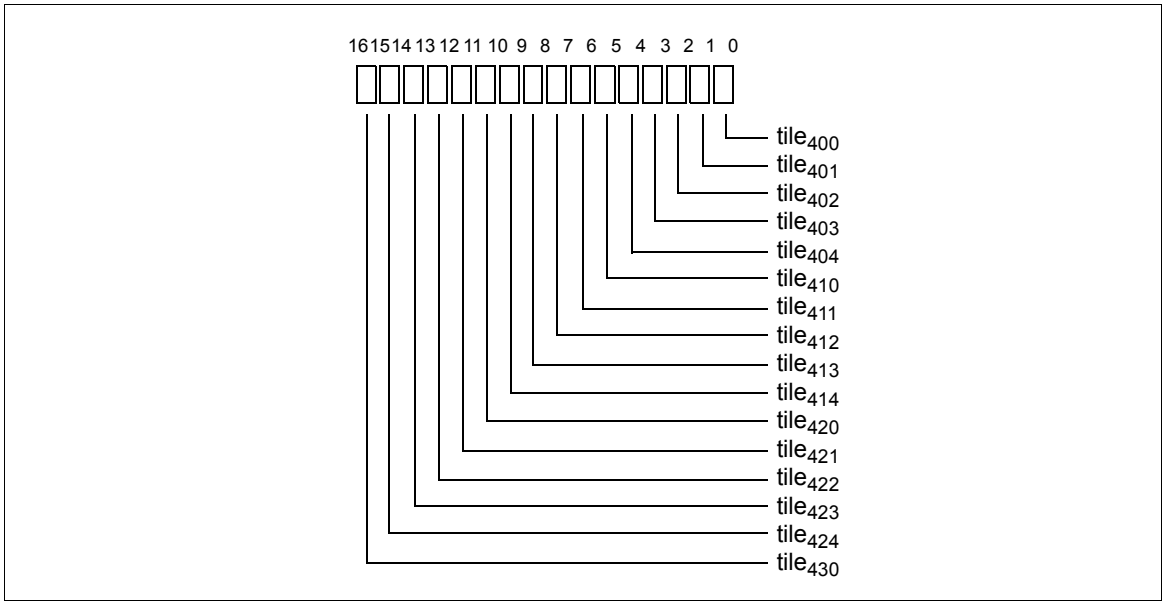

Figure 129 Contribution to event of XZP group in Veto List

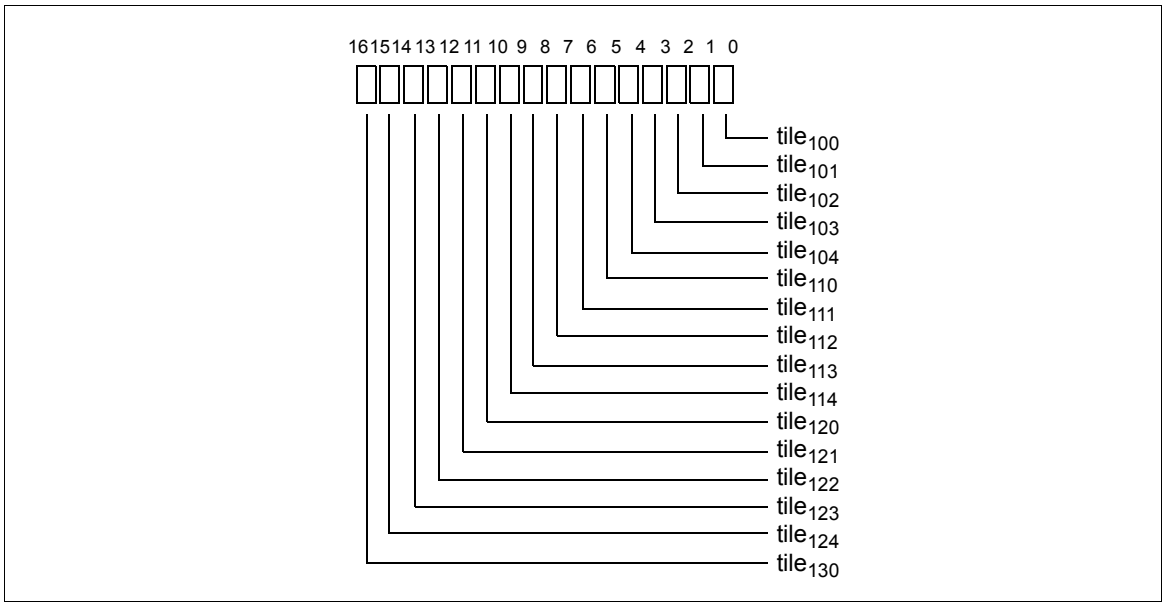

Figure 130 Contribution to event of YZM group in Veto List

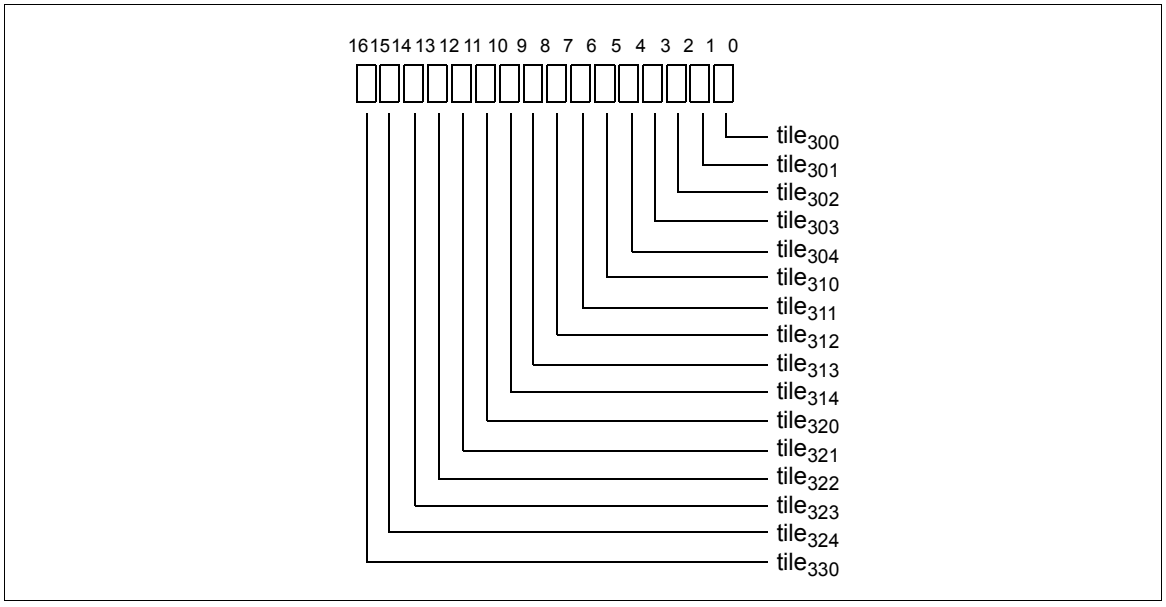

Figure 131 Contribution to event of YZP group in Veto List

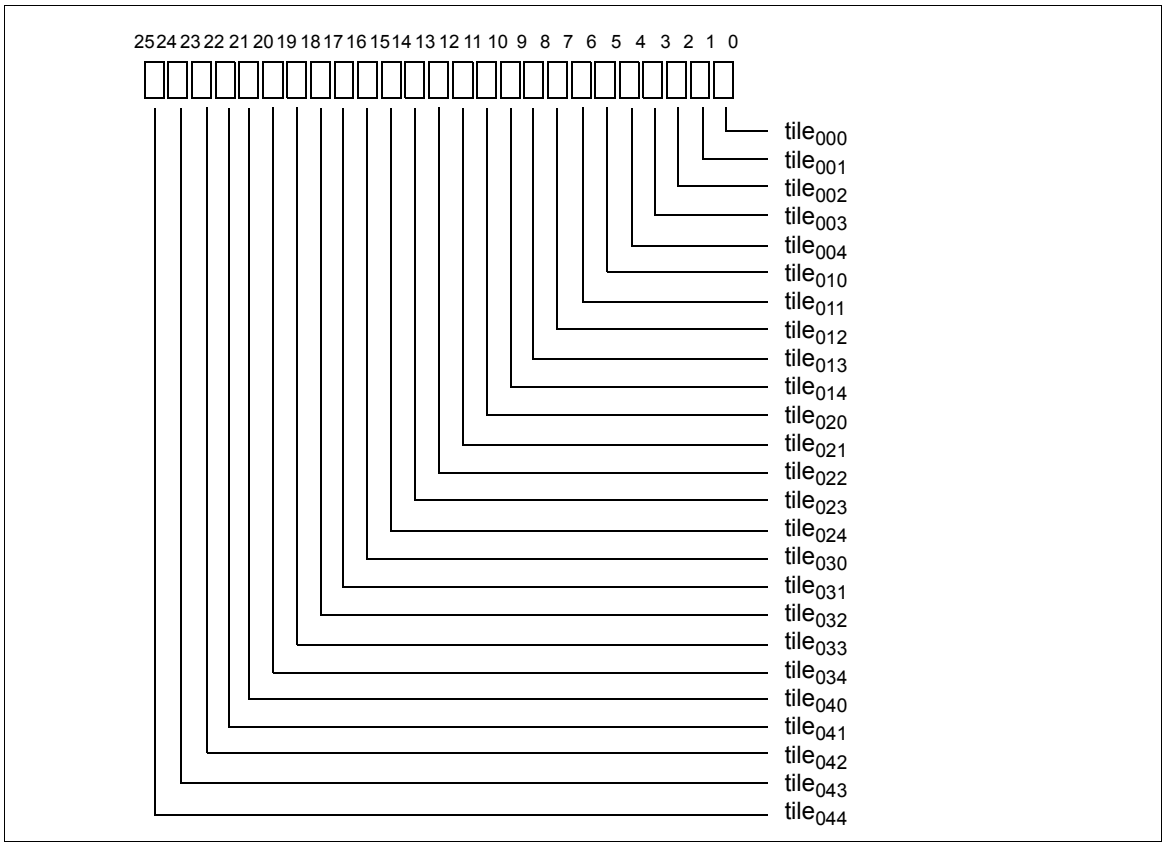

Figure 132 Contribution to event of XY group in Veto List

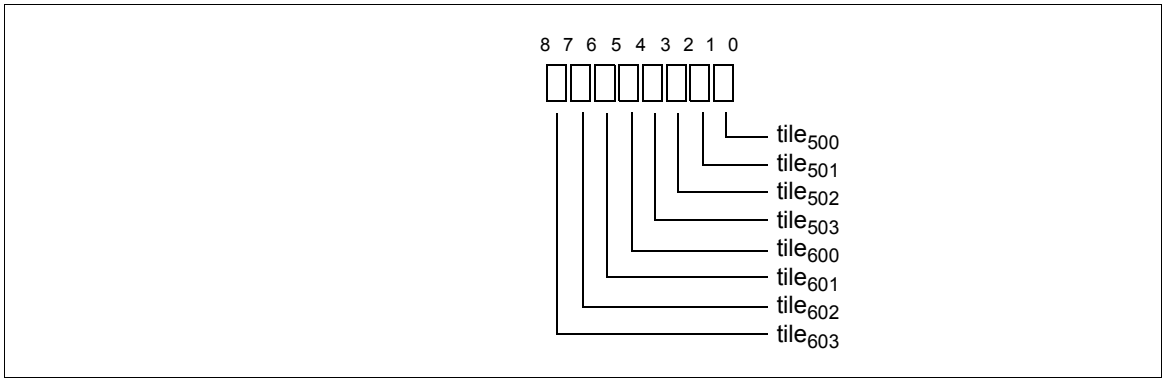

Figure 133 Contribution to event of RBN group in Veto List

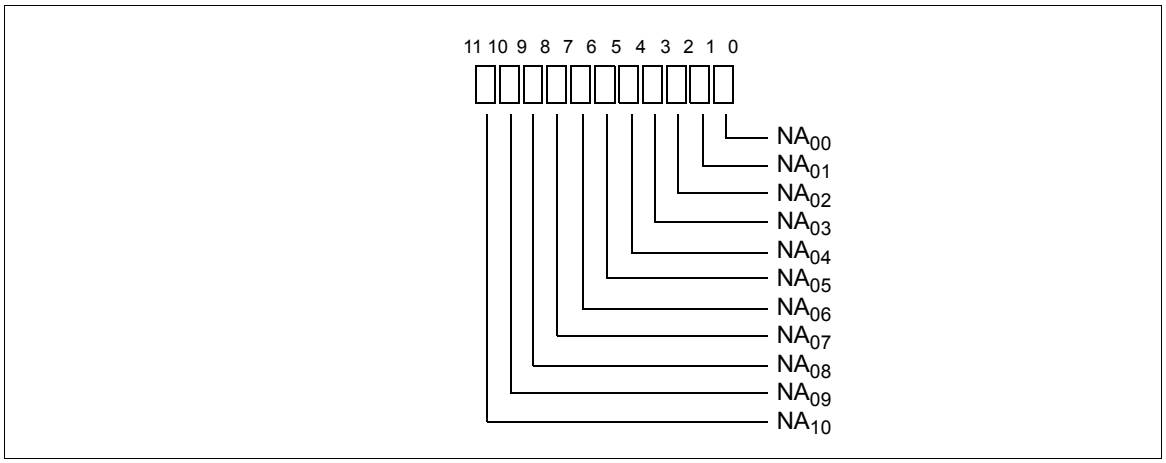

Figure 134 Contribution to event of NA group in Veto List

4.10 Sampled livetime

The GEM maintains a tally of the livetime in the LAT. (See Section 2.6.) At event hold time this counter is sampled and its value recorded in the contribution illustrated in Figure 135. Note: One count corresponds to 50 nano-seconds.

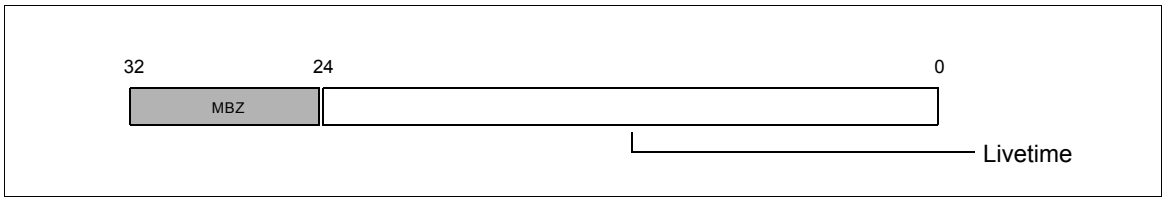

Figure 135 Livetime contribution to event

4.11 Sampled prescaled count

The GEM maintains a counter of the number of times a window was turned, but its corresponding prescaler did *not* expire. (See Section 2.6.) At event hold time, this counter is sampled and its value recorded in the contribution illustrated in Figure 136:

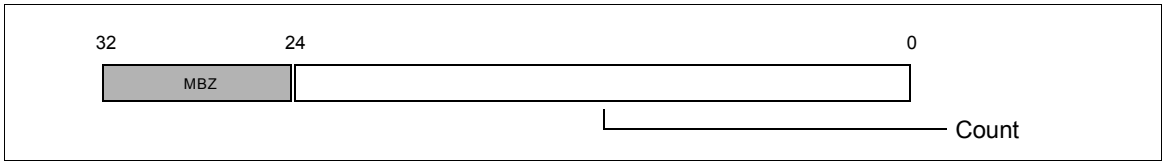

Figure 136 Prescaled counter contribution to event

4.12 Sampled discarded count

The GEM maintains a count of the number of times a window was turned and its corresponding prescaler expired; however, the LAT was *busy*. (See Section 2.6.) At event hold time, this counter is sampled and its value recorded in the contribution illustrated in Figure 137:

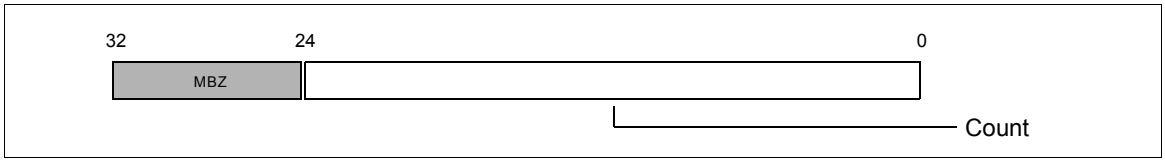

Figure 137 Discard counter contribution to event

4.13 Condition arrival times

For six of the most interesting conditions the GEM measures the relative arrival times of the conditions which occurred while the window corresponding to the event was open. Time is measured in units of system clock (nominally 50 nanoseconds). As the window open time has maximum range of 5-bits, this contribution contains six 5-bit fields. Each field corresponds to the arrival time relative to the window opening. Consequently, the condition (or conditions) which opened the window will have a time of *zero* (0), and the maximum arrival time corresponds to a value of *thirty* (30) for a field. There are two circumstances under which a value of *thirty-one* (31) will be recorded:

i. The corresponding field for the condition is *not* marked TRUE in the condition summary.

ii. The corresponding field for the condition *is* marked TRUE in the condition summary, however, the condition was already present when the window was opened.

The structure of this contribution is illustrated in Figure 138:

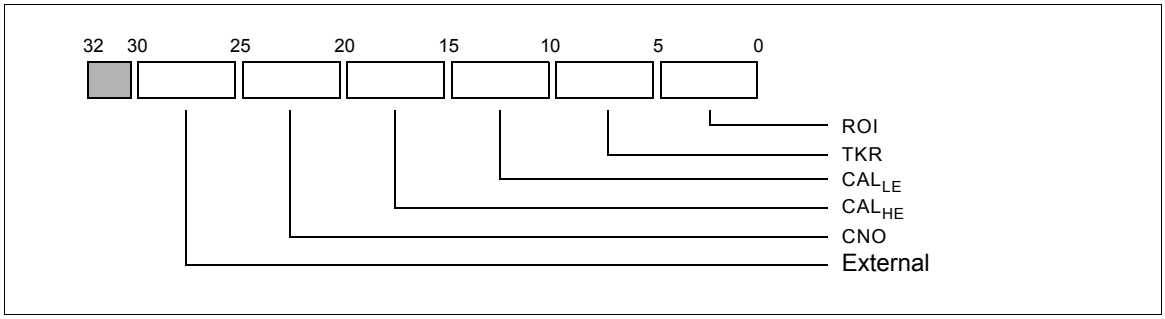

Figure 138 Sent counter contribution to event

4.14 Trigger Time

This contribution to the event consists of the sampled state of the GEM's timebase when the window corresponding to the event was *closed*. The timebase is free-running, 25-bit counter, incrementing at the system clock rate (20 MHZ). Thus, one count in the timebase corresponds to 50 nanoseconds. Note that this counter simply continues on overflow. Therefore, this contribution nominally represents the count of system clock ticks from the moment the counter was reset to the event being declared. The format of the contribution is illustrated in Figure 139.

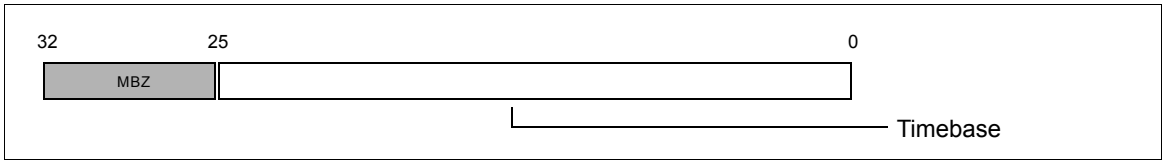

Figure 139 Trigger time

4.15 Sampled 1-PPS time

The GEM maintains two registers which are updated upon the arrival of the spacecraft's 1-PPS signal:

seconds: This register counts the number of arrived 1-PPS signals since the GEM was reset; thus, it increments each time a 1-PPS signal arrives. As the 1-PPS signal arrives once per second, this register is nominally a count of the number of seconds since the GEM was reset. This register is 7 bits wide. Note that this counter simply continues on overflow.

1-PPS time: Each time the 1-PPS signal arrives, the GEM's timebase (discussed in Section 4.14) is sampled and saved in this register. Thus, this register contains the time, in system clock ticks (50 nano-seconds), of the last arrived 1-PPS signal. As the timebase is a 25-bit counter, this register is 25 bits wide.

This contribution is simply a sample of these two registers when the window corresponding to the event was *closed*. The format of the contribution is illustrated in Figure 140.

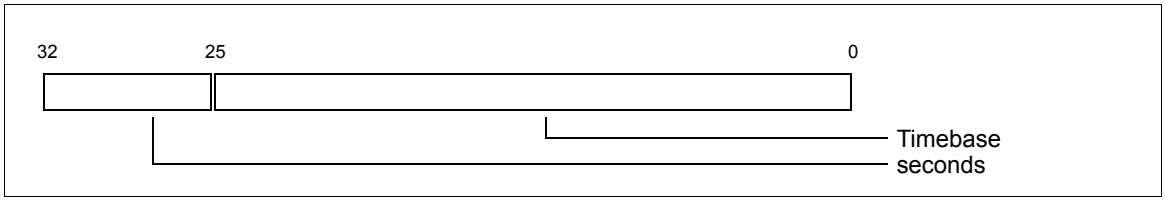

Figure 140 1-PPS timing

4.16 Delta Event time

The GEM maintains a count of the elapsed time from event *n*-1 to event *n*. Time is counted in system clocks, where one tick is nominally 50 nanoseconds. At event hold time, this counter is sampled, its value is recorded in the contribution illustrated in Figure 141, and it is reset to *zero*. This is a 16-bit saturating counter, which once its maximum value is reached will stop incrementing until reset. The counter is set to zero and begins incrementing whenever the GEM is reset. Consequently, the *first* event contribution emitted by the GEM after a reset will contain the elapsed time from GEM *reset* to declaration of the first event.

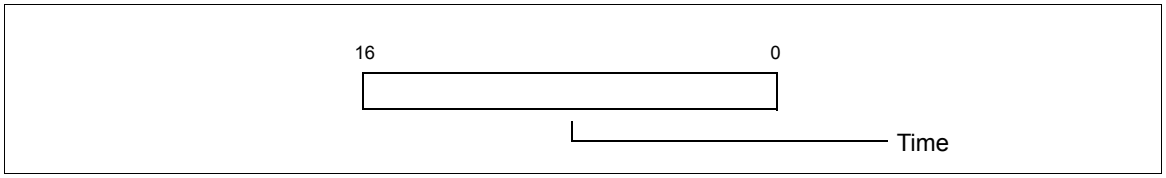

Figure 141 Delta event time contribution to event

4.17 Delta window open time

The GEM maintains a count of the elapsed time from window-open *n*-1 to window-open *n*. Time is counted in system clocks, where one tick is nominally 50 nanoseconds. At event hold time (which corresponds to window open *n*), this counter is sampled and its value is recorded in the contribution illustrated in Figure 142. In short, this contribution measures the elapsed time from the event to the last time a window was opened. This is a 16-bit saturating counter, which once its maximum value is reached will stop incrementing until reset. The counter is set to zero and begins incrementing whenever the GEM is reset. Consequently, the *first* event contribution emitted by the GEM after a reset will contain the elapsed time from GEM *reset* to declaration of the first event.

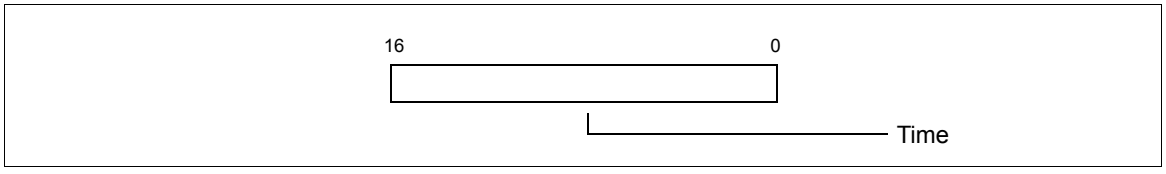

Figure 142 Delta window open time contribution to event

Appendix A
Tile Mapping

Figure A.1 illustrates the geometry of the ACD and the 12 FREE boards shown surrounding the tiles. The representation of a FREE board identifies its name (for example, `4LA`, or `2LB`) and lists its electronics channels and the tile which is mapped to the channel. By ACD convention, *A* fibres (as represented on a FREE board) have a tile prefix of `0` and *B* fibres have a tile prefix of `1`. For example, the *A*-side of the tile numbered `0022` is mapped to channel `15` of FREE board `2LA`. The same tile, but its *B*-side, is numbered as `0122` and is mapped to channel `2` of FREE board `2LB`. The unassigned channels are shown as *blank*. While these channels aren't mapped to tiles, the GEM pairs up the unassigned channels and `or`s them together, and they go into the trigger decision[1] just as if they were backed by a tile.

1. Suitably masked off by the software responsible for configuring the GEM.

Figure A.1 ACD tile mapping

For best channel utilization of the LVDS receivers used by the GEM, the NA (not assigned) tiles are interleaved in the configuration interface for both input enables and tower assignment. (See Section 2.8 and Section 2.9.3.) A tabular view of the geometry as illustrated by Figure A.1 is presented in Table A.1:

Table A.1 Correspondence between tiles, FREE boards, and tile number

Tile Name	PMT "A"		PMT "B"		Tile Number[1]
	Board	Chnl	Board	Chnl	
000	2LA	6	2LB	11	00
001	2LA	12	2LB	5	01
002	2LA	17	2LB	0	02
003	2RA	6	2RB	11	03
004	2RA	11	2RB	6	04

Table A.1 Correspondence between tiles, FREE boards, and tile number

Tile Name	PMT "A"		PMT "B"		Tile Number[1]
	Board	Chnl	Board	Chnl	
010	2LA	7	2LB	10	05
011	2LA	13	2LB	4	06
012	2RA	4	2RB	13	07
013	2RA	5	2RB	12	08
014	2RA	10	2RB	7	09
020	2LA	8	2LB	9	0A
021	2LA	14	2LB	3	0B
022	2LA	15	2LB	2	0C
023	4LA	15	4LB	2	0D
024	4LA	9	4LB	8	0E
030	4RA	10	4RB	7	0F
031	4RA	5	4RB	12	10
032	4RA	4	4RB	13	11
033	4LA	14	4LB	3	12
034	4LA	8	4LB	9	13
040	4RA	11	4RB	6	14
041	4RA	6	4RB	11	15
042	4LA	17	4LB	0	16
043	4LA	13	4LB	4	17
044	4LA	7	4LB	10	18
NA2	4RA	13	4RB	16	19
NA3	4RA	16	2LB	16	1A
100	2LA	1	1RB	3	1B
101	1LA	6	1RB	7	1C
102	1LA	9	1RB	8	1D
103	1LA	10	1RB	11	1E
104	1LA	14	4RB	1	1F
110	2LA	0	1RB	2	20
111	1LA	5	1RB	6	21

Table A.1 Correspondence between tiles, FREE boards, and tile number

Tile Name	PMT "A"		PMT "B"		Tile Number[1]
	Board	Chnl	Board	Chnl	
112	1LA	8	1RB	9	22
113	1LA	11	1RB	12	23
114	1LA	15	4RB	0	24
120	1LA	0	1RB	1	25
121	1LA	4	1RB	5	26
122	1LA	7	1RB	10	27
123	1LA	12	1RB	13	28
124	1LA	16	1RB	17	29
130	1LA	17	1RB	0	2A
NA4	1LA	1	1RB	16	2B
NA5	1LA	3	1RB	14	2C
200	2LA	5	2LB	12	2D
201	2LA	11	2LB	6	2E
202	2RA	3	2RB	14	2F
203	2RA	7	2RB	10	30
204	2RA	12	2RB	5	31
210	2LA	3	2LB	14	32
211	2LA	10	2LB	7	33
212	2RA	2	2RB	15	34
213	2RA	8	2RB	9	35
214	2RA	14	2RB	3	36
220	2LA	2	2LB	15	37
221	2LA	9	2LB	8	38
222	2RA	0	2RB	17	39
223	2RA	9	2RB	8	3A
224	2RA	15	2RB	2	3B
230	2RA	17	2LB	17	3C
NA6	2LA	16	2LB	13	3D
NA7	2RA	13	2RB	16	3E

Table A.1 Correspondence between tiles, FREE boards, and tile number

Tile Name	PMT "A"		PMT "B"		Tile Number[1]
	Board	Chnl	Board	Chnl	
300	3LA	14	2RB	1	3F
301	3LA	10	3RB	11	40
302	3LA	9	3RB	8	41
303	3LA	6	3RB	7	42
304	4LA	1	3RB	3	43
310	3LA	15	2RB	0	44
311	3LA	11	3RB	12	45
312	3LA	8	3RB	9	46
313	3LA	5	3RB	6	47
314	4LA	0	3RB	2	48
320	3LA	16	3RB	17	49
321	3LA	12	3RB	13	4A
322	3LA	7	3RB	10	4B
323	3LA	4	3RB	5	4C
324	3LA	0	3RB	1	4D
330	3LA	17	3RB	0	4E
NA8	3LA	3	3RB	14	4F
NA9	3LA	1	3RB	16	50
400	4RA	12	4RB	5	51
401	4RA	7	4RB	10	52
402	4RA	3	4RB	14	53
403	4LA	12	4LB	5	54
404	4LA	6	4LB	11	55
410	4RA	14	4RB	3	56
411	4RA	8	4RB	9	57
412	4RA	2	4RB	15	58
413	4LA	11	4LB	6	59
414	4LA	5	4LB	12	5A
420	4RA	15	4RB	2	5B

Table A.1 Correspondence between tiles, FREE boards, and tile number

Tile Name	PMT "A"		PMT "B"		Tile Number[1]
	Board	Chnl	Board	Chnl	
421	4RA	9	4RB	8	5C
422	4RA	0	4RB	17	5D
423	4LA	10	4LB	7	5E
424	4LA	3	4LB	14	5F
430	4RA	17	4LB	17	60
NA0	4LA	2	4LB	15	61
NA1	4LA	16	4LB	16	62
500	3LA	13	1RB	4	63
501	3LA	2	1RB	15	64
502	1LA	2	3RB	15	65
503	1LA	13	3RB	4	66
600	2LA	4	4RB	4	67
601	4RA	1	2LB	1	68
602	2RA	1	4LB	1	69
603	4LA	4	2RB	4	6A
NA10	2RA	16	4LB	13	6B

1. In hexadecimal