The GLAST experiment at SLAC

The Command/Response Unit

Electronics group

Programming ICD specification

Document Version:	3.0
Document Issue:	4
Document Edition:	English
Document Status:	Under release control
Document ID:	LAT-TD-01547
Document Date:	July 14, 2004

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

Abstract

A conceptual design of the CRU (Command/Response Unit). This module is designed to facilitate communications on the Command/Response fabric. This fabric, or communications mesh, is used by the LAT for internal command, monitoring, and control.

Hardware compatibility

This document assumes the following hardware revisions:

CRU: Version *0*, Board Revision *2*, *Engineering* Type

Intended audience

This document is intended principally as a guide for the *developer* and *users* of the CRU. Users include:

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

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

Conventions used in this document

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

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

3 "ACD Electronics Module - Programming ICD specification," Michael Huffer, LAT-TD-00639.

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 Command/Response Unit Programming ICD specification		
	Version:	3.0		
	Issue:	4		
	Edition:	English		
	ID:	LAT-TD-01547		
	Status:	Under release control		
	Created:	February 9, 2002		
	Date:	July 14, 2004		
	Access:	V:\GLAST\Electronics\Design Documents\CRU\v3.0\Frontmatter.fm		
	Keywords:	Command/Response Unit CRU		
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 Command/Response Unit Programming ICD specification		
ID:	LAT-TD-01547		
Version	**Issue**	**Date**	**Reason for change**
1.2	1	3/18/2003	Initial draft, for internal comment
1.3	1	5/01/2003	Changed title and added NASA verbiage to conform to both Gunther's and NASA's wishes for CDR. Incorporated comments from Gunther, JJ, and Tony
2.0	1	6/20/2003	Some many changes to both document and design, that one should consider this a fresh start.
2.1	1	3/05/2004	Updated fonts.
2.2	1	4/06/2004	Corrected typos.
3.0	1	5/09/2004	This version reflects the "as built" CRU. The changes are too numerous to enumerate. It's best to just reread the Register chapter.
3.0	3	5/17/2004	Corrected minor typos. Sent document for sign-off.
3.0	4	7/14/04	Updated references and PDF TOC.

Table of Contents

List of Figures

List of Tables

Chapter 1
Principles of operation

1.1 The CRU and the Command/Response Fabric

The principal function of the CRU (Command/Response Unit) is the distribution of the various signals between the nodes of the Command/Response fabric. (See [1].) From the perspective of the CRU, the fabric's nodes can be partitioned into two sets: those which are capable of commanding other nodes (***masters***) and those which may only respond to commands (***slaves***). Although the fabric supports up to six masters, only one of these masters may command at any one time. Such a master is referred to as the designated *commander*. Any node which does not, or cannot, command is called a *responder*. The CRU's principal functions are three-fold:

a. Synthesise and distribute (fan-out) the system clock (`sysclk`) to all nodes on the fabric.

b. Distribute (fan-out) command packets from commander to responder.

c. Distribute (fan-in) response packets from responder to commander.

Consequently, one may view the CRU as the "glue" binding together the nodes of the fabric. The components of the CRU are organized along with the AEM (ACD Electronics Module), EBM (Event Builder Module), and GEM (GLT Electronics Module) on a single Printed Circuit Board (PCB) called the DAQ board. The DAQ board resides in the GASU *box* as described in [2]. 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 CRU on the primary DAQ board is called the *primary* CRU and the CRU on the redundant DAQ board, the *redundant* CRU. Only one of the two CRUs can operate at any one time; it can, however, be either the primary or the redundant unit. A diagram expressing the CRU and its interfaces is illustrated in Figure 1.

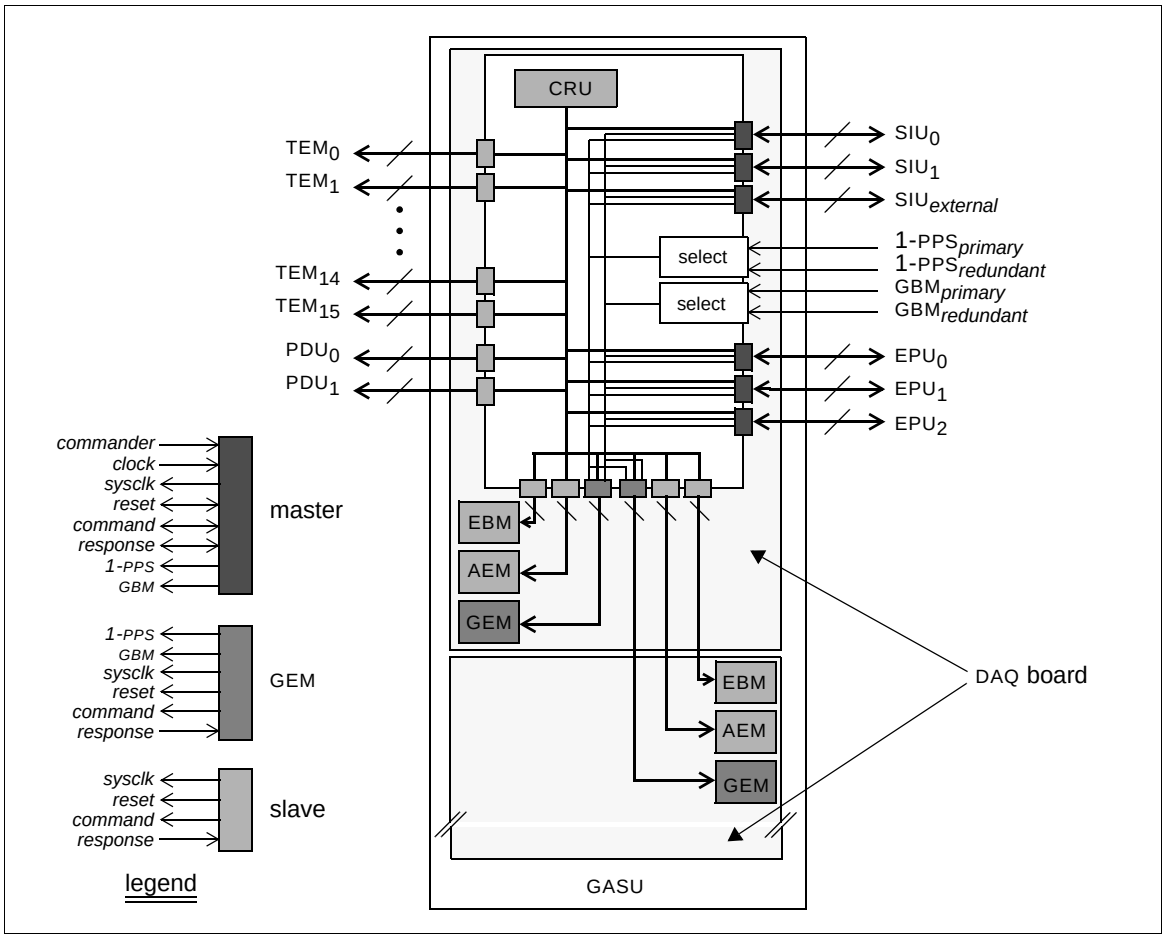

Figure 1 Functional interface for the CRU

The diagram illustrates that the GASU provides an interface to each and every node of the Command/Response fabric. Note that the signal interface for a node varies as a function of whether the interface is for a master or a slave.[1] It is convenient to speak of the two sets of EBM, AEM, and GEM contained within the GASU as the CRU's *internal* nodes and the nodes outside the GASU as the CRU's *external* nodes. Signals from and to external nodes are carried differentially using LVDS. The external nodes include:

- 16 TEMs (Tower Electronics Module)
- 2 PDUs (Power Distribution Unit)
- 3 SIUs (System Interface Unit)
- 3 EPUs (Event Processor Unit)

The relationship between node interface and its corresponding physical connector on the GASU is described in [2]. The CRU also receives two sets of timing signals from the spacecraft: the 1-PPS (one Pulse-Per-Second) and the GBM (Gamma-Ray Burst Module). One set is

1. The GEM is special case, as it also receives the external timing signals. However, in this document it will be treated as if it were a slave on the fabric.

redundant, and the CRU is used to select the one appropriate set of signals to be distributed to the nodes of the fabric.

1.1.1 Slave interface

The slave interface defines the four common Command/Response wires:

sysclk: Carried on this wire is the system clock (nominally 20 MHZ). The CRU *transmits* this signal.

reset: Carried on this wire is the hardware reset signal. The CRU *transmits* this signal.

command: Carried on this wire are encoded LATp packets corresponding to a command directed from the commander to the corresponding node. The CRU *transmits* this signal.

response: Carried on this wire are encoded LATp packets corresponding to the response to a command directed from the commander to the corresponding node. The CRU *receives* this signal.

1.1.2 Master interface

The master interface defines the same four Command/Response wires as the slave interface. (See Section 1.1.1.) However, three of these four wires (***reset***, ***command***, and ***response***) can be either transmitters or receivers. The sense of these wires is determined by the boolean value of the commander wire as illustrated in Figure 2:

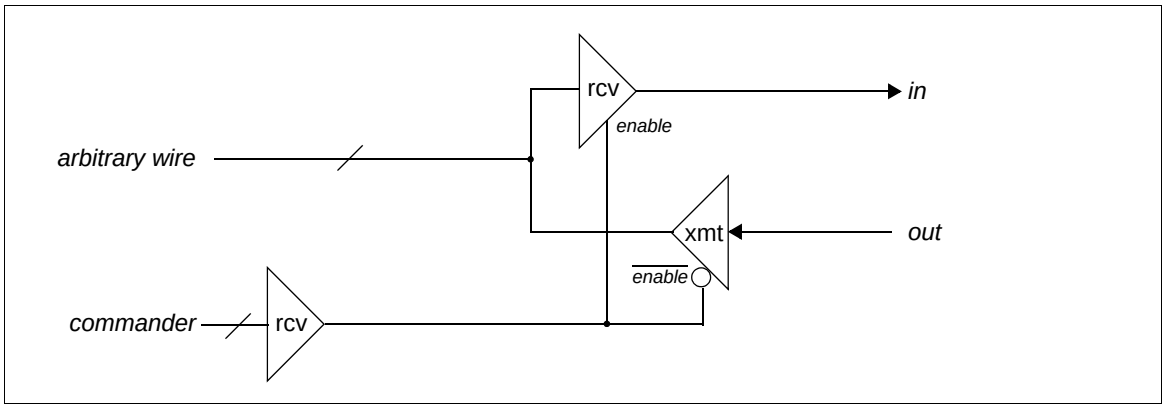

Figure 2 Multiplexing a master wire

The complete list of wires for the master interface include:

commander: This signal determines whether the node corresponding to the interface is the designated commander. The signal carried on this wire is a constant boolean level. A TRUE level specifies the node as a commander. A FALSE level specifies the node is a responder. The ***clock***, ***reset***, ***command***, and ***response*** interface are affected by the value of the signal on this wire. The CRU *receives* this signal.

clock: Carried on this wire is the clock (nominally 40 MHZ) which the CRU will use to synthesize the LAT system clock (sysclk). This signal is masked with the ***commander*** signal described above. The CRU *receives* this signal.

sysclk: Carried on this wire is the system clock (nominally 20 MHZ). The CRU *transmits* this signal.

reset: Carried on this wire is the hardware reset signal. Depending on the state of the ***commander*** signal (described above) the CRU either transmits or receives this signal. If the ***commander*** signal is TRUE, the CRU *receives* this signal. If the ***commander*** signal is FALSE, the CRU *transmits* this signal.

command: Carried on this wire are encoded LATp packets corresponding to a command directed from the commander to the corresponding node. Depending on the state of the ***commander*** signal (described above) the CRU either transmits or receives this signal. If the ***commander*** signal is TRUE, the CRU *receives* this signal. If the ***commander*** signal is FALSE, the CRU *transmits* this signal.

response: Carried on this wire are encoded LATp packets corresponding to the response to a command directed from the commander to the corresponding node. Depending on the state of the ***commander*** signal (described above) the CRU either transmits or receives this signal. If the ***commander*** signal is TRUE, the CRU *transmits* this signal. If the ***commander*** signal is FALSE, the CRU *receives* this signal.

1-PPS: Carried on this wire is the One-Pulse-Per-Second signal. This signal originated from the spacecraft. The CRU *transmits* this signal.

GBM: Carried on this wire is the Gamma-Ray-Burst Monitor signal. This signal originated from the spacecraft. The CRU *transmits* this signal.

1.1.3 The CRU as a node on the fabric

The CRU is passive with respect to the packets and signals which flow through it. In other words, commander and responder are unaware of its presence when communicating together. However, the fabric has a significant amount of built-in redundancy which must be "trimmed" before the fabric can be used for communication.[1] This trimming is a responsibility of the CRU and consequently, the CRU requires a certain amount of "one-time" configuration. The CRU's configuration is maintained in a series of registers in a fashion entirely analogous with all other nodes on the fabric. Therefore, in order to access these registers, the CRU is *itself* a slave node on the fabric. The functional differences in interface between the CRU and other nodes on the fabric are both trivial and straight-forward, but are noted here:

1. Trimming is discussed in detail in [1].

- The CRU's (LATp) address is both fixed and immutable. Assignment of node addressees for the nodes of the fabric is dynamic and requires configuration of the CRU. (See [1].) Consequently, its address must be fixed. (The address of the CRU is `1E`, hexadecimal.)
- The CRU's (LATp) interface signals may *not* be masked. (See, for example, Section 2.3.2.) Again, this allows the CRU to always have the capacity to perform necessary one-time initialization of the fabric.

The slave interface is represented as the "CRU" box within Figure 1 and as the "Command/Response Engine" box within Figure 3. An enumeration and discussion of the CRU's configuration registers is found within Chapter 2. The encoding necessary to access these registers is found in Chapter 3.

1.2 Implementation model

A block diagram of the CRU is illustrated in Figure 3. The CRU has both *Front*-End and *Back*-End interfaces. The Front-End services both master and external timing signals, the Back-End, slave related signals. Front-ends and Back-ends consist of *Fan-In* and *Fan-Out* pairs, each pair identified with the processing of one particular kind of signal. A Fan-In block receives and reduces the same type of signal from *n* different sources (in some cases, suitably masked) to one signal. A Fan-Out block takes one signal and multiples and transmits that signal (in some cases, suitably masked) to *m* destinations. In this fashion Fan-In and Fan-Out blocks may be connected together to route *n* sources to *m* destinations. Note that depending on origin, signals may be routed from Front-End to Back-End, or from Back-End to Front-End, or even from Front-End to Front-End. For example, *commands* that are fanned-in by the Front-End are fanned-out by the Back-End. *Responses* are fanned-in by the Back-End and fanned-out to the Front-End.

Physically, the CRU consists of a single FPGA to handle the combinatorial logic and a miscellaneous number of commercial LVDS transmitters/receivers, to process control, input and output signals.

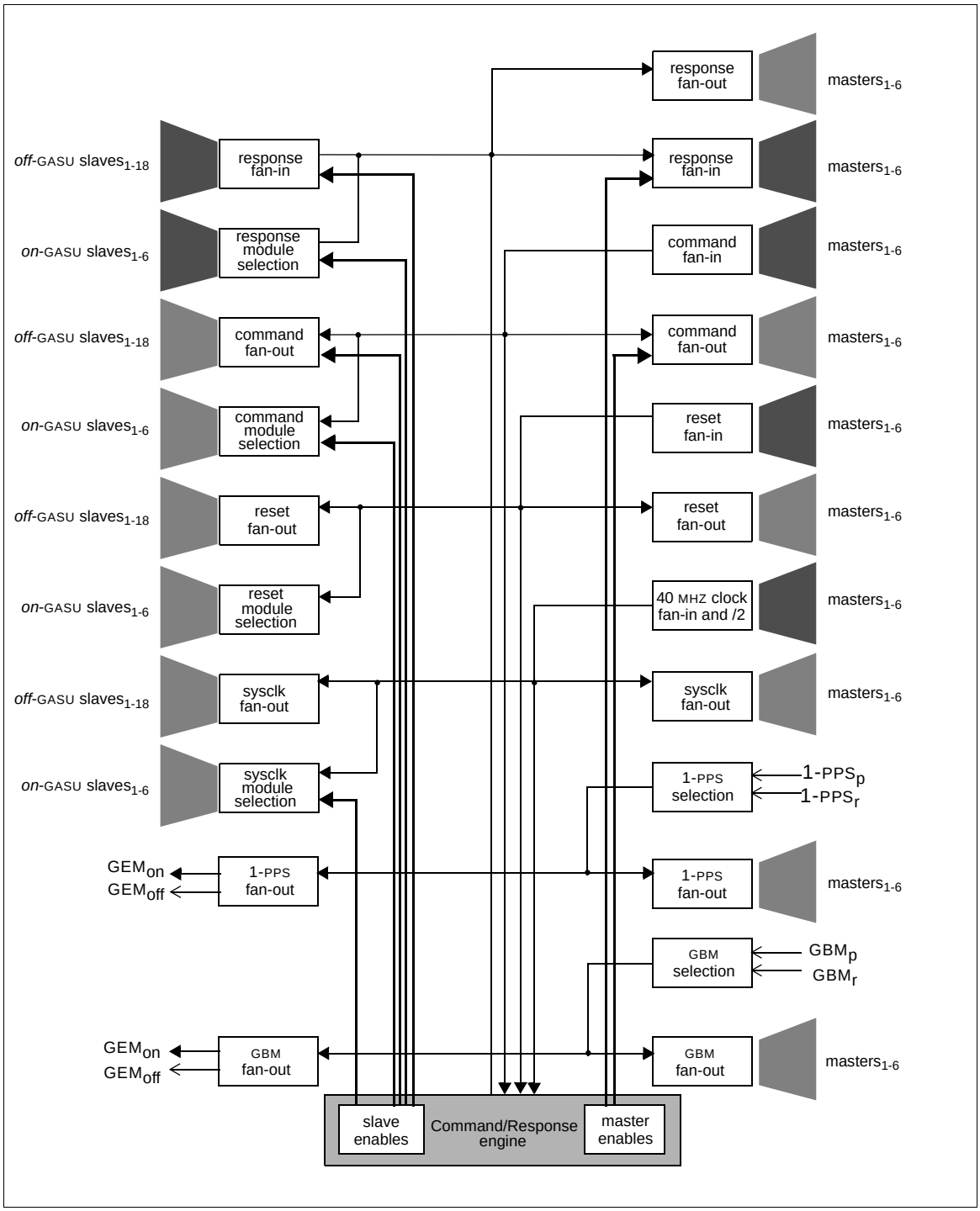

Figure 3 Block diagram of the CRU

1.2.1 Clock Processing

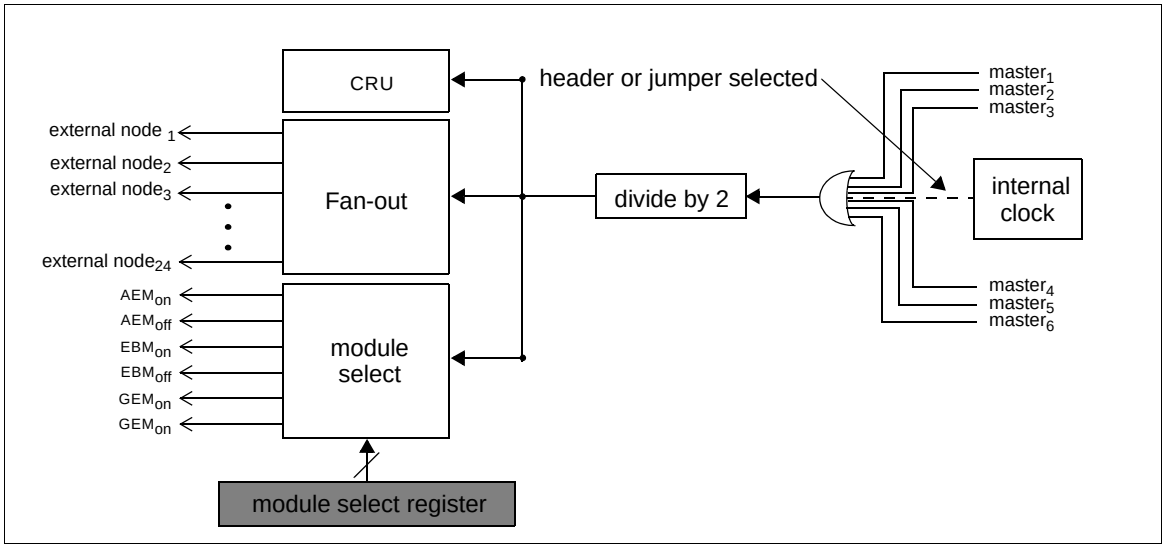

Figure 4 Fan-in and Fan-out of system clock (`sysclk`)

1.2.2 Reset processing

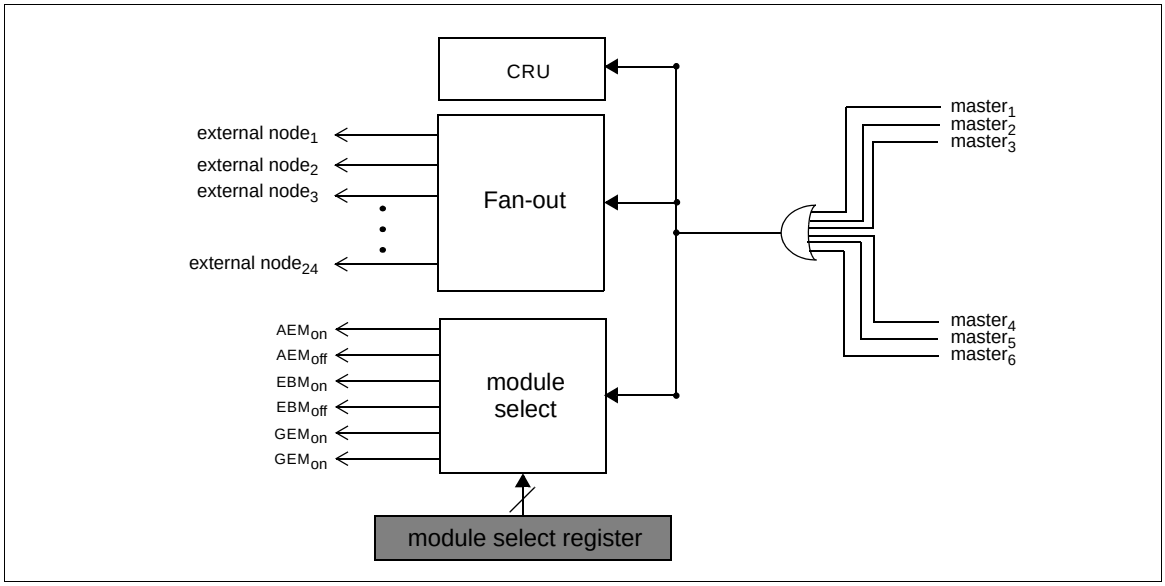

Figure 5 Fan-in and Fan-out of reset signals

1.2.3 Command Processing

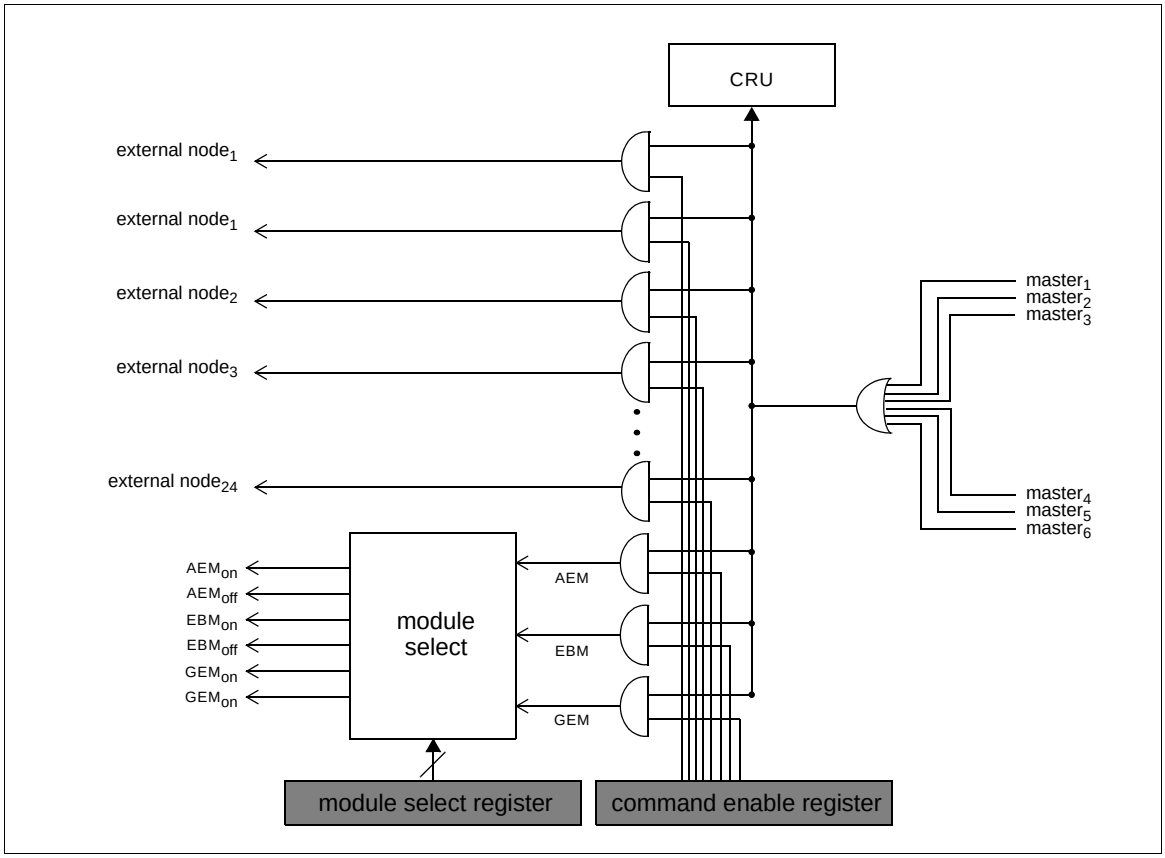

Figure 6 Fan-in and Fan-out of command signals

1.2.4 Response Processing

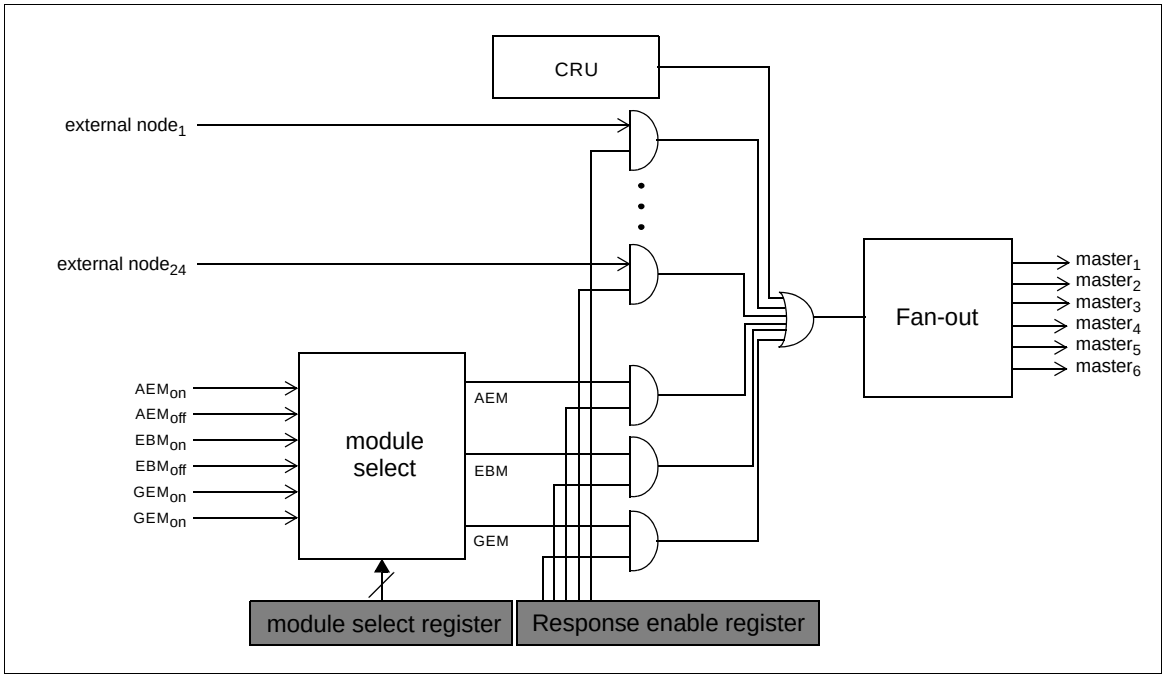

Figure 7 Fanning-in and fanning-out the response signals

1.2.5 1-PPS and GBM selection

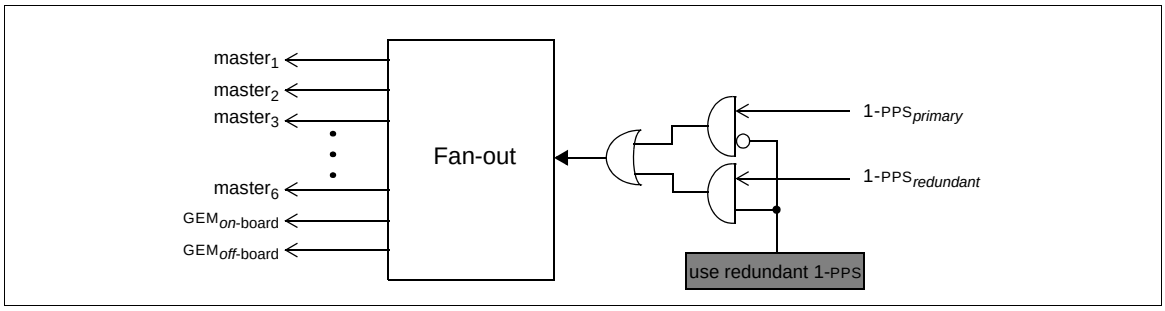

Figure 8 Forming the Periodic Open Window Signal

Chapter 2
Registers

2.1 Introduction

The CRU contains a series of *registers* for configuration and management. These registers are contained in one block called the CRU Controller. 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 CRU 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 Controller registers

This section incorporates all the registers whose configuration has a global affect over all the functional blocks of the CRU. All registers are read/write.

Table 4 The registers of the controller block

Name	Address	Description
CONFIGURATION	00	Configuration and setup
COMMAND_ENABLE	01	List of nodes whose *command* signal is masked
RESPONSE_ENABLE	02	List of nodes whose *response* signal is masked
CR_STATISTICS	03	Accumulated Command/Response statistics
ACD_STARTUP_CLOCK	04	Status of power-up clock train for FREE boards
TEST_I/O	05	External test connector Input/Output
Total	**6**	

2.3.1 Configuration

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

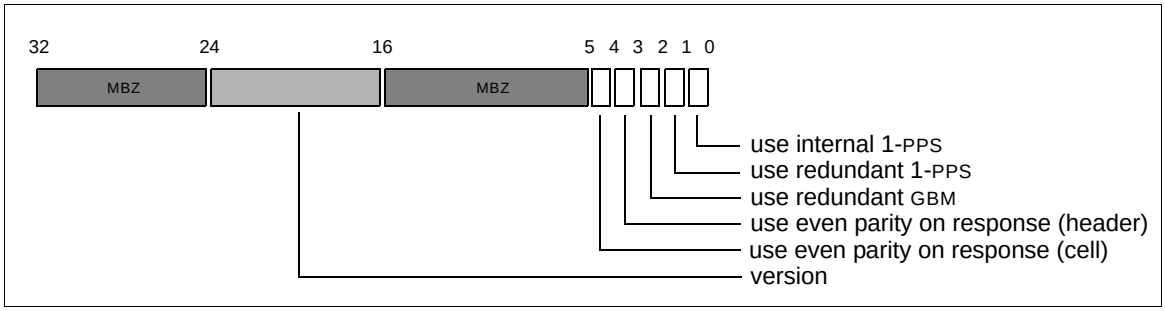

Figure 9 Configuration register

use internal 1-PPS: Determines whether the Once-Pulse-Per-Second (1-PPS) signal will be synthesized by the CRU rather then the spacecraft. If the field is *clear*, the *1*-PPS is expected to be provided by the spacecraft. If the field is *set*, the 1-PPS signal is synthesized by the CRU. Note: If this field is set, the value of the field ***use redundant 1-PPS*** is ignored.

use redundant 1-PPS: Determines which one of the two Once-Pulse-Per-Second (1-PPS) signals from the spacecraft will serve as the source of the CRU's 1-PPS signal. For reliability, there are two identical 1-PPS signals: by convention, labelled the *primary* and *redundant* 1-PPS. If the field is *clear*, the *primary 1*-PPS signal is used by the CRU. If the field is *set*, the *redundant* 1-PPS signal is used by the CRU.

use redundant GBM: Determines which one of the two Gamma-Ray-Burst-Monitor (GBM) signals from the spacecraft will serve as the source of the CRU's GBM SIGNAL. For reliability, there are two identical GBM signals: by convention, labelled the *primary* and *redundant* GBM. If the field is *clear*, the *primary* GBM SIGNAL is used by the CRU. If the field is *set*, the *redundant* GBM signal is used by the CRU.

use even parity on response (header): Determines whether the *header* parity generated by the CRU 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 on response (cell): Determines whether the *cell* parity generated by the CRU 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*.

version: Specifies the hardware revision level of the CRU. The structure of this field is defined in Figure 10. 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 the module was implemented and how it was intended to be used. The values for this field are defined in Table 5.

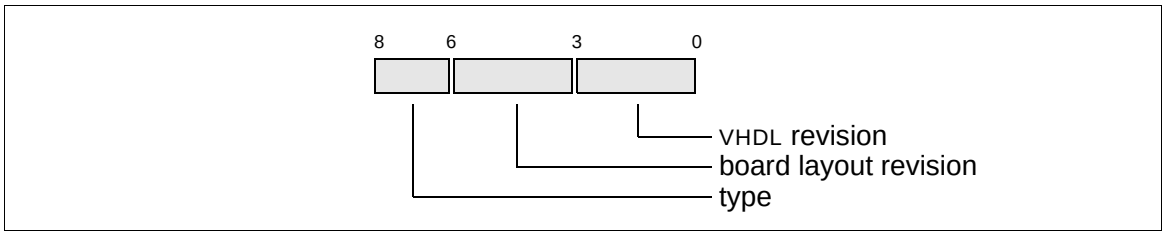

Figure 10 Structure of revision register or field

Table 5 Usage of the type field of the revision register

Value[1]	description
00	Software emulation/engineering model
01	Engineering model
10	Qualification model
11	Flight model

1. In binary

2.3.2 Command enable register

The CRU is responsible for fanning-in and then fanning-out the fabric ***command*** signal received from any one of the fabric's potential responders. (See Section 1.2.3.) The COMMAND ENABLE register specifies a bit-list of the responders whose command signal the CRU can fan-out. The structure of this register is illustrated in Figure 11. If the bit at a specified offset is *set*, the command signal for the corresponding responder will be fanned-out (enabled). If the bit is *clear*, the command signal for the corresponding responder is *not* fanned-out (masked).

This register is used to establish the addresses of the nodes of the Command/Response fabric. (See [1].)

Note: The command line used by the CRU *itself is by default always enabled.*

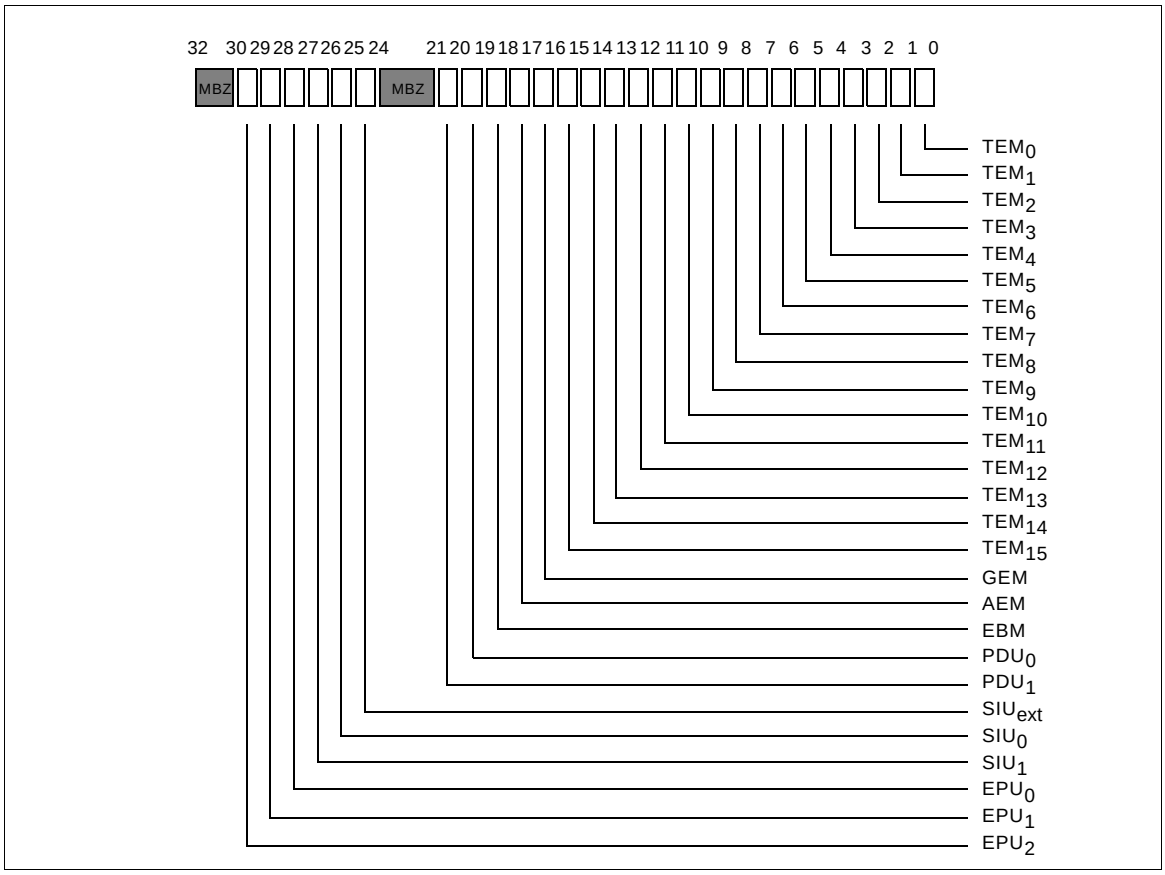

Figure 11 Command enable register

2.3.3 Response enable register

The CRU is responsible for fanning-in and then fanning-out the fabric ***response*** signal received from any one of the fabric's potential responders. (See Section 1.2.4.) The ***response enable*** register specifies a bit-list of the responders whose response signal the CRU will fan-in. The structure of this register is illustrated in Figure 12. If the bit at a specified offset is *set*, the response signal for the corresponding responder is fanned-in (enabled). If the bit is *clear*, the response signal for the corresponding responder is *not* fanned-in (masked). As the fan-in for

the response signal from all the responders is simply ORed (as discussed in Section 1.2.4), this register may be used to mask-off "chattering," or noisy responders.

Note: The response line used by the CRU itself is by default always enabled.

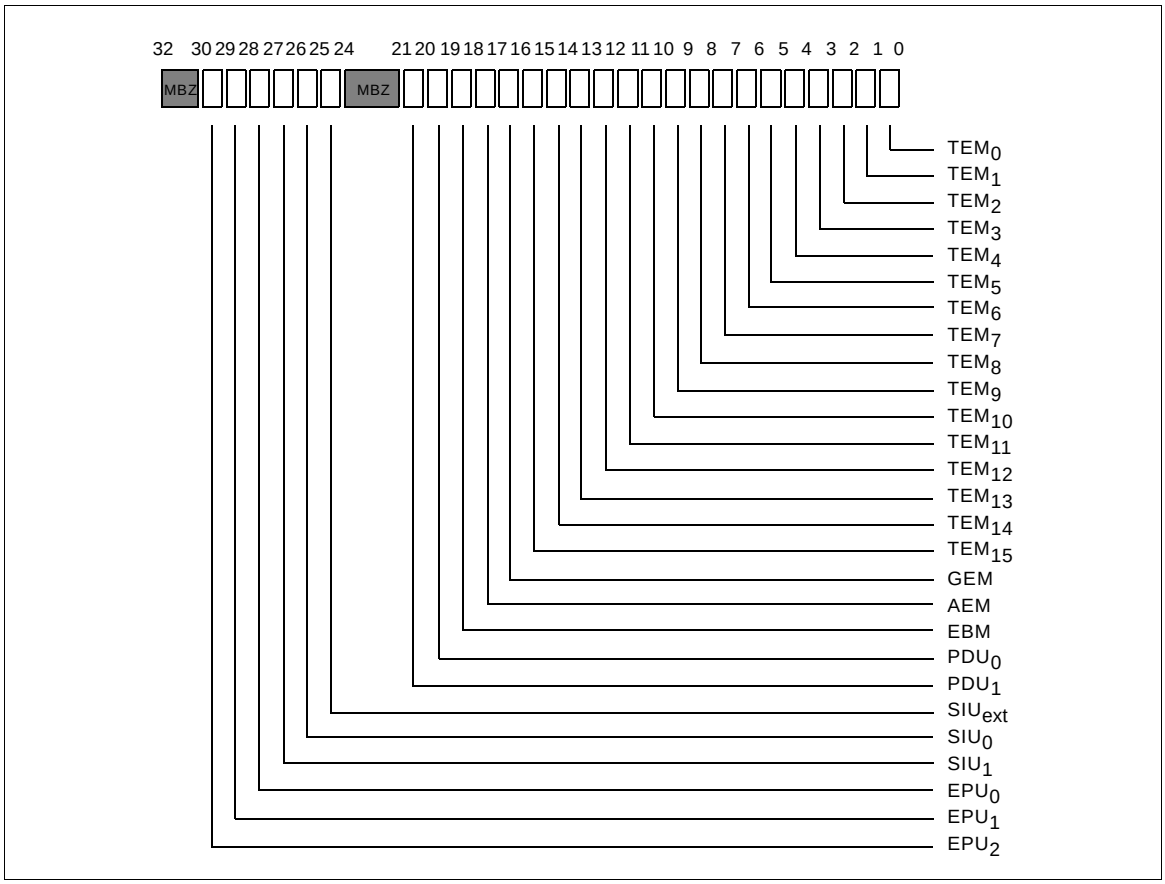

Figure 12 Response enables register

2.3.4 Command/Response statistics register

The CRU is a node on the Command/Response fabric. As such it is obligated (see [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 13. Note, that when the register is written, the value to be written is ignored and the register is set to *zero*.

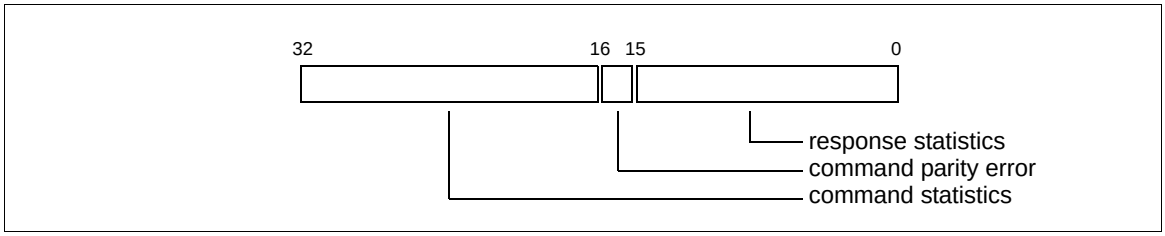

Figure 13 The Command/Response statistics register

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

command parity error: This field is set when the CRU access descriptor (see Figure 17) of an incoming command has a parity error.

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

2.3.5 ACD startup clock

The GARC (GLAST ACD Readout Controller) is the ASIC providing the interface between the ACD and GASU. The ACD uses twelve GARCs, one for each FREE-board. In order to overcome an error in the GARC's fail-over logic, it is necessary, when powering on a FREE board to provide a small number of clock pulses. The frequency of this pulse is 1.25 MHZ and its duration is one second[1]. This register reflects whether or not this clock is being applied and has the structure illustrated within Figure 14. Each bit offset corresponds to the clock state for its corresponding FREE board. If the field is *set*, the clock is operating. If the field is *clear*, the clock is not operating. The clock train is triggered by applying power to the appropriate FREE board using a register on the AEM. (See [3].)

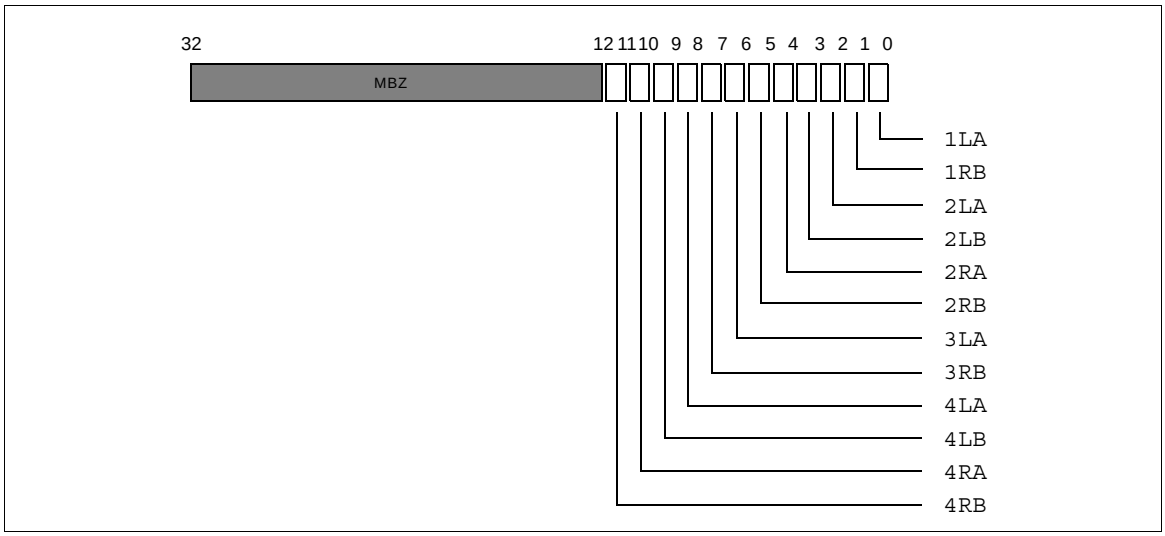

Figure 14 ACD startup clock status register

1. This value is currently very generous and will be turned down when the appropriate value is discovered. Note: While these clocks are being generated, communication with the GARC is impossible.

2.3.6 Test I/O register

The test connector (see [2]) of the GASU provides two signals whose usage depends on application. These signals are not available in flight and are intended to be used only to interface ground support equipment to the dataflow system. For example, they are used to coordinate activity between the FES (Front-End Simulator) and dataflow system on the testbed. (See xxx.) One set of wires is used to *transmit* an external LVDS signal and the other set of wires is used to *receive* an LVDS signal. This register provides an interface to these two sets of wires. Both these signals use the dataflow convention of negative TRUE logic. The "In" field reflects the state of the receive signal and is *set*, if logically TRUE and *clear*, if logically FALSE. Note: this field is Read-Only. The "Out" field is to be described.

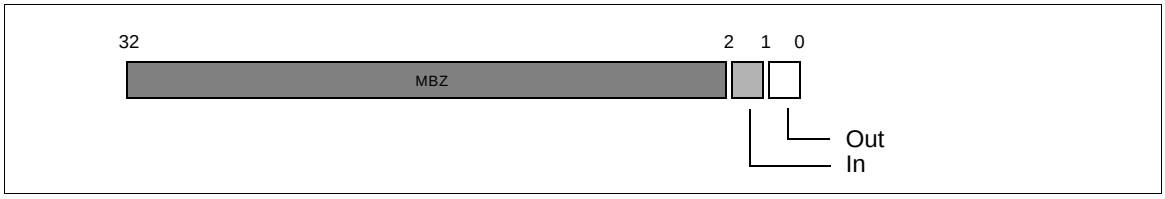

Figure 15 Test I/O register

Chapter 3
Commanding

3.1 Overview

This chapter describes the remote protocol necessary to access both the registers[1] and functional blocks of the CRU. It follows the Command/Response Protocol discussed in [1]. The registers of the CRU are organized into a hierarchy of only one block as illustrated in Figure 16.

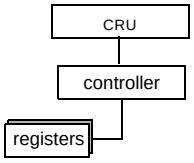

Figure 16 Hierarchy of target types

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 CRU's access descriptor

Directly following the LATp header of any received command packet is a fixed size, 16-bit structure, called the CRU's ***access descriptor***. This descriptor is a parameterization of the access rules required to address the functional blocks and registers of the CRU. Its structure is illustrated in Figure 17:

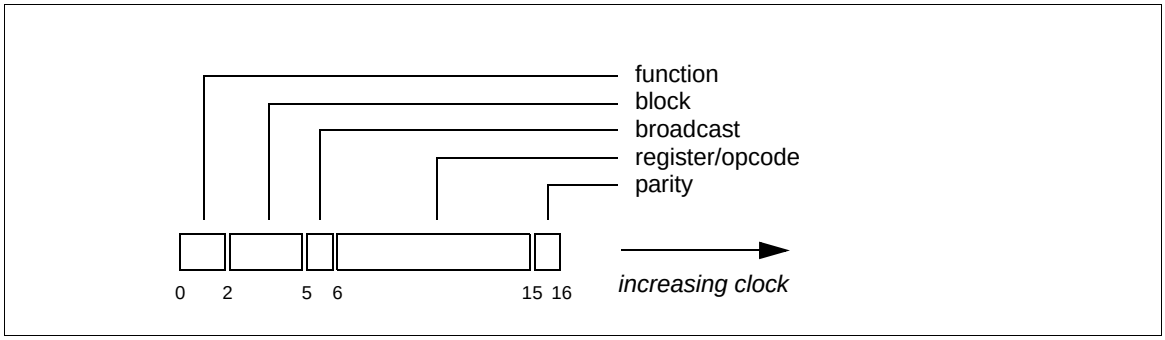

Figure 17 CRU access descriptor

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 enumerations for this field are described in [1].

block: This field enumerates which of the blocks of the CRU are to be accessed. The correspondence between block type and number is enumerated in Table 6.

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 6 Block numbers of the CRU

Name	number
`controller`	0

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 7 The controller's dataless commands

Name	Opcode	Description
`NOP`	0	No operation
`RESET`	1	Hard reset of the CRU
Total	**2**	

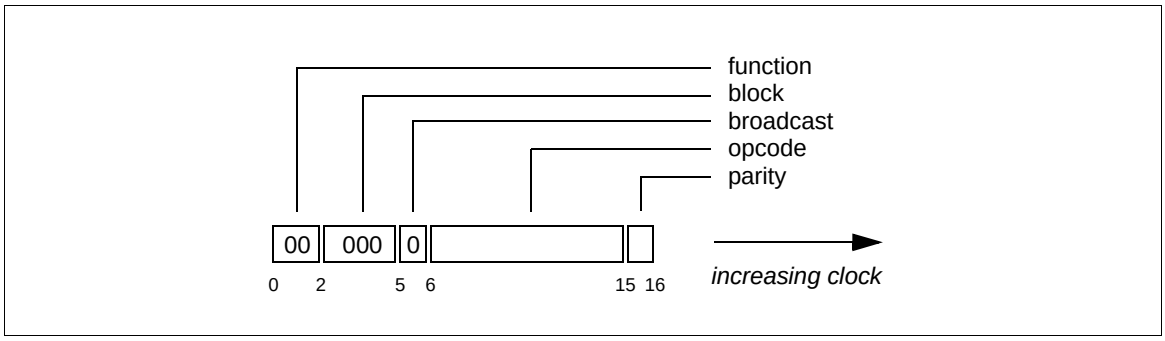

Figure 18 Access descriptor for the controller's dataless commands

`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

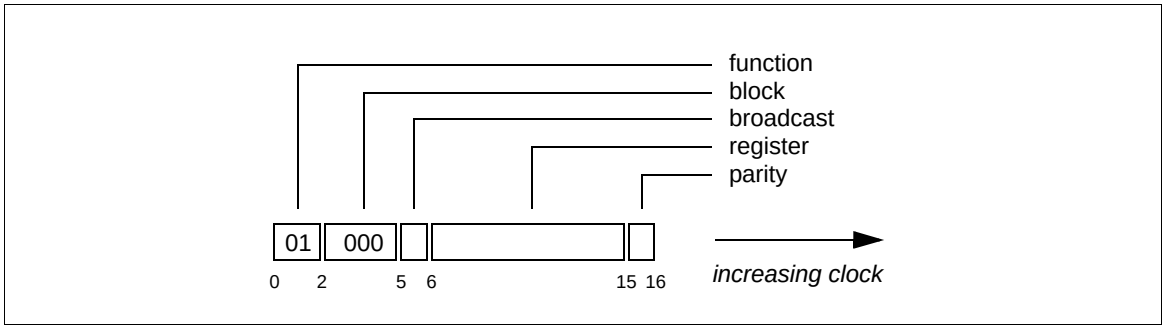

Figure 19 Access descriptor for the controller's register load commands

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 20. As a `Load` function does not require a response, the ***respond*** field of the packet is set to *false*.

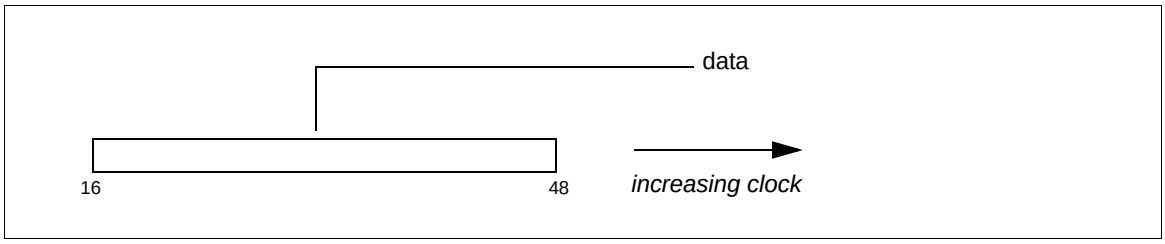

Figure 20 Payload for the controller's register load commands

3.3.3 Read commands

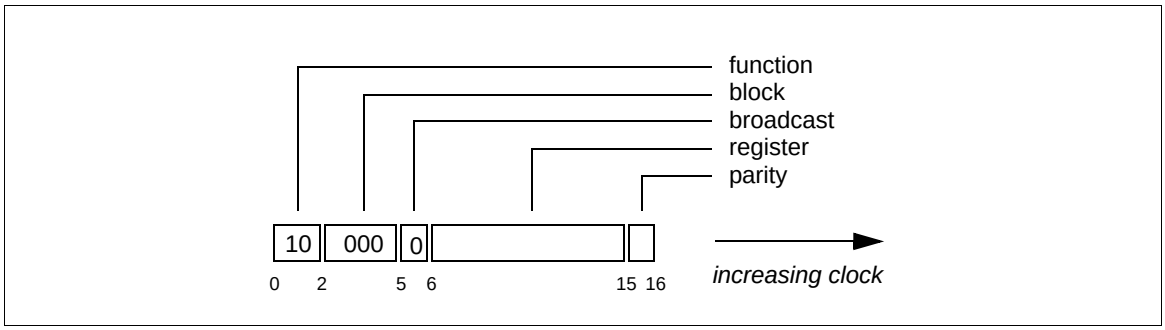

Figure 21 Access descriptor for the controller's register read commands

`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 22.

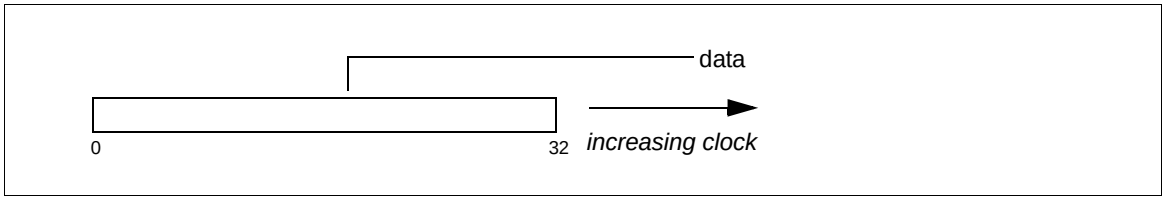

Figure 22 Response to a register read command of the controller