LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	**LIGO-T980017-00 - Cxx**	3/6/98

Data Acquisition System
Reflected Memory Network Design

R. Bork

Distribution of this draft

This is an internal working note
of the LIGO Project..

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (818) 395-2129
Fax (818) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

1 ABSTRACT

This document describes the design for the Data Acquisition System (DAQS) reflected memory networks. The intent of this document is to define the network architecture and the appropriate interfaces, such that software development can begin on the various LIGO components which must connect to this system.

2 REFERENCE DOCUMENTS

- Data Acquisition System Final Design
- Global Diagnostics System Preliminary Design

3 OVERVIEW

3.1. Network Architecture

The interface to the DAQS is in three primary forms:

- Analog signals directly into ADC units of the DAQS and digitized at 16384Hz or 2048Hz. DAQS VME systems which provide this interface are called Data Collection Units (DCU).
- Direct digital via CDS control networks and EPICS channel access (Network Data Collection Unit). These signals are acquired at 1Hz and referred to as ‘Slow Data Channels’.
- Directly on the DAQS reflected memory network from Interferometer Sensing and Control (ISC) processors.
- Directly on the DAQS reflected memory network from components of the Global Diagnostics System (GDS).

To then accomodate the collection and distribution of data from these various sources, the DAQS provides two reflected memory networks, as shown in Figure 1:DAQS Reflected Memory Network. The DCU crates contain ADC units provided by the DAQS. The Input Optics, ASC and LSC crates (denoted as IFO ISC) contain the processors and digitizers for the ISC systems. These units are provided as part of the ISC subsystems, with the reflected memory being the primary interface into the DAQS.

Additional units shown are:

- DAQS Controller (DAQSC): Provides oversight and synchronization functions for the DAQS.
- FrameBuilders: Collect data, format the data into LIGO/VIRGO standard frame formats, and store the data to disk.
- GDS Search: GDS component which generates triggers and captures associated event data.
- Excite Engine: GDS component which controls IFO diagnostic tests.

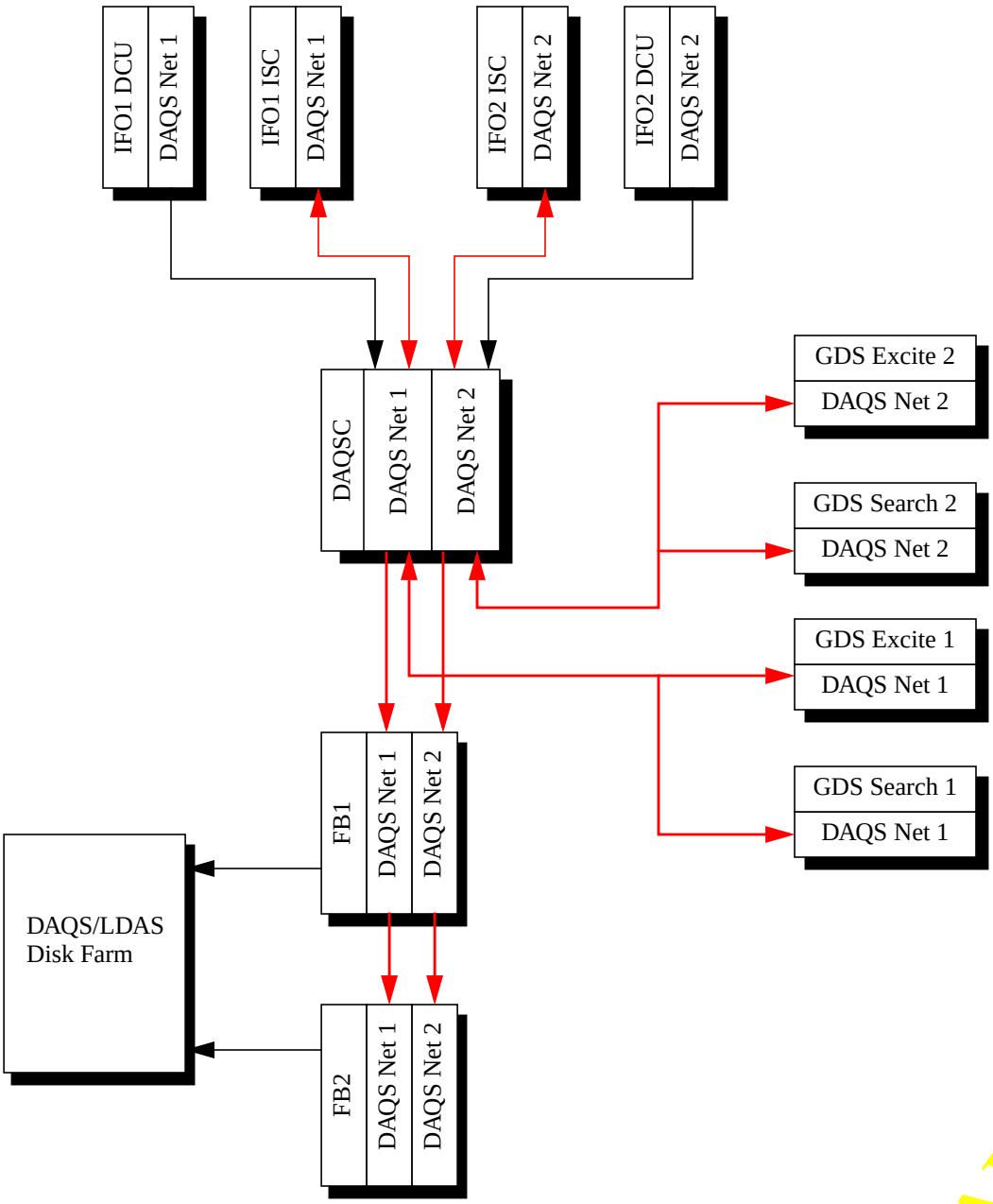

Figure 1: DAQS Reflected Memory Network

3.2. Reflected Memory Space Allocations

The DAQS reflected memory networks allocate space for both data collected by the DAQS and processed data and ISC excitation data generated by the GDS. The allocation and general layout is shown in Figure 2:Memory Architecture. The upper half of this memory contains all of the data collected from the DAQS DCU and ISC processors. The lower half contains the GDS generated data. The upper half is controlled by the DAQS and is described further in this document. The lower half is to be controlled in a similar manner by the GDS and will be further described in separate GDS documentation.

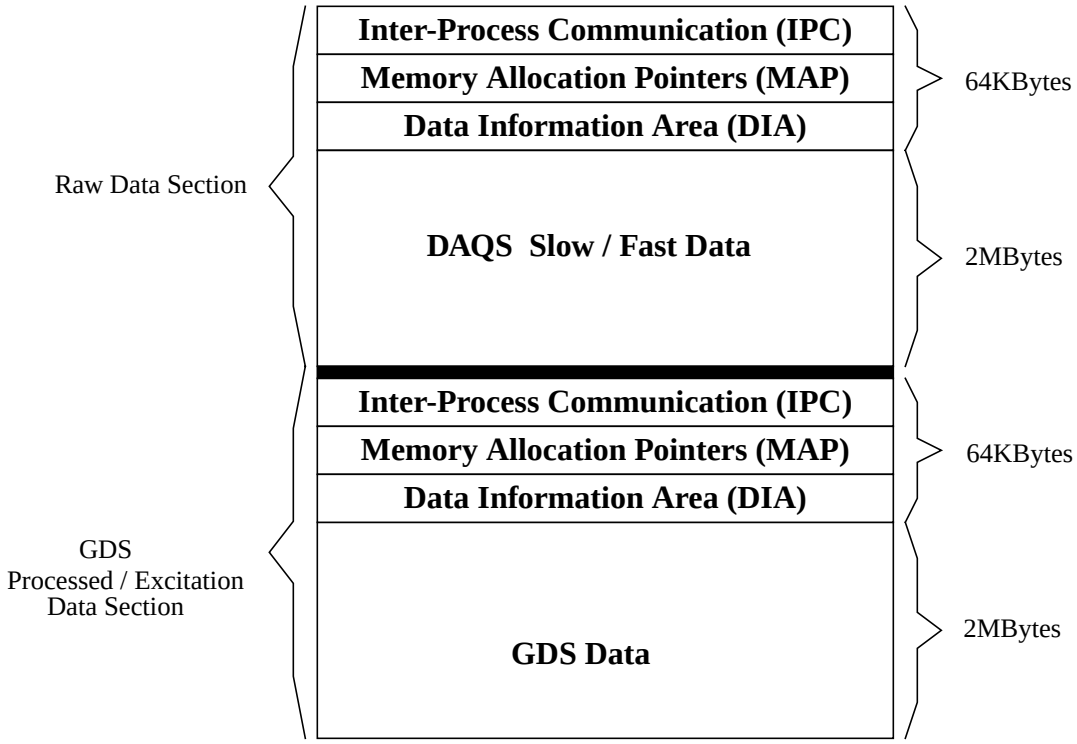

Figure 2: Memory Architecture

4 DAQS MEMORY DESIGN

The DAQS portion of the reflected memory networks each contain 2MByte of memory and are to be partitioned as shown in Figure 3:Reflected Memory Structure and defined as follows:

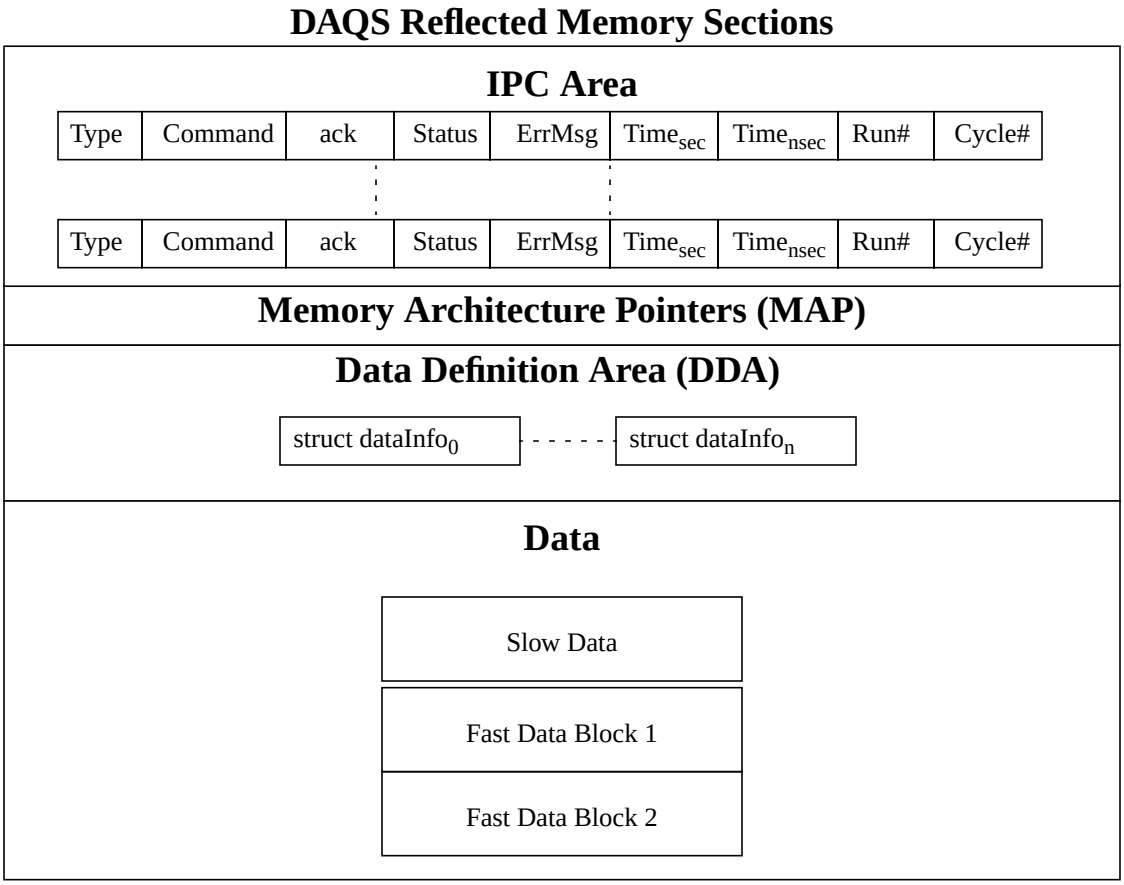

Figure 3: Reflected Memory Structure

- InterProcess Communication (IPC): This section allows for direct communication of the various DAQS processors through the reflected memory. Once the DAQS is powered up and booted, this is the primary communication path within the DAQS.
- Memory Architecture Pointers (MAP): This block describes the arrangement of the reflected memory, thereby allowing other software to locate data within the reflected memory.
- Data Definition Area: Contains information about the data being acquired, such as names, rates, locations, etc.
- Data Area: Contains the actual data being acquired. Note that for fast data, two blocks are assigned. These blocks each contain 1/16 second of data, and are written to alternately by the DCUs.

4.1. Data Collection Overview

To better understand the reflected memory description which follows, a brief overview of the data collection process is described as the following sequence:

- On power up, the DAQS controller will pull up the latest system configuration file and setup the MAP and DIA sections of the reflected memory and indicate (via its status field within the IPC area of reflected memory) that a valid memory configuration is present and usable by any processes on the network.
- As DCUs power up, they will initialize, indicate that they are on and ready, and then await commands from the DAQS controller.
- When the DAQS controller sees a DCU join the network, it will automatically issue configuration and start acquisition commands to those DCU. When they are ready, the DCUs will begin data acquisition on the next 1Hz time marker from their GPS receiver.
- If, during operation, the DAQS controller receives an operator command to switch to a new configuration, it will reload the MAP and DIA and then issue synchronized commands for all DCU to switch to the new configuration.

While the DAQS controller and the DCUs constantly acquire data and enter it into the reflected memory whenever they are powered up, the FrameBuilders wait for operator commands and frame configuration information prior to pulling data from the reflected memory and storing it to disk. This process, along with additional DAQS operation, is further discussed in the DAQS software design document.

4.2. DCU Startup Information

On initial boot and load of DCU software, the following information is available to each DAQS processor (TBD if info is a definition in the dcu code itself or as part of a startup script). This information is necessary for the processors to interpret data in the reflected memory.

- The identification number of this processor (0-20).
- Base address of the reflected memory IPC area.
- Base address of the reflected memory MAP area.
- Number and clock rates for ADC modules

4.3. IPC Area

The IPC area contains thirty daqIPC structures, as described in Table 1, “struct daqIPC,” on page 8. Definition of the command, status, and errMsg fields are TBD. The DAQS controller information is always contained in the first IPC structure in both DAQS networks.

Table 1: struct daqIPC

Data class	*Variable Name*	*Descriptor & Comments*
int	type	Type of processor: DAQS controller, FrameBuilder, DCU, NDCU

Table 1: struct daqIPC

Data class	*Variable Name*	*Descriptor & Comments*
int	command	Command to DCU's from DAQS Controller
int	ack	Command acknowlege field
int	status	DCU status information to DAQS Controller
int	errMsg	Error messages from DCU/NDCU to Controller
int	timeSec	Time in seconds of last data block transferred
int	timeNsec	Time in nanoseconds of last data block transferred
int	run	Number of present data run (set by DAQS controller at each new configuration)
int	cycle	Counter of data blocks transferred; reset to 0 at start of each run and at cycle = cycle % 1048576

4.3.1. TYPE Field

The Type field indicates to DAQS processors the function which they are to perform. The defined types are shown in the following table.

Table 2: Type Field Definition

Value	*Descriptor & Comments*
1	DAQS Controller
2	FrameBuilder
3	Generic DCU
4	NDCU

4.3.2. Command Field

The Command field is used by the DAQS controller to send commands to the DCUs. This field is comprised of bits, as indicated in the following table.

Table 3: Command Field Definition

Bit	*Name*	*Descriptor & Comments*
0	Configure	0 = Data configuration has not changed. 1 = New data configuration is available in the reflected memory.
1	Stop/Run	0 = Stop acquisition on next 1Hz GPS; 1 = Start acquisition on next 1Hz GPS

4.3.3. Status Field

The Status field communicates status information from each DCU back to the DAQS controller. This is a bit pattern as defined in the following table.

Table 4: Status Field Definition

Bit	*Name*	*Descriptor & Comments*
0	Configure	0 = Not configured to acquire data 1 = Configured to acquire data.
1	Stop/Run	0 = DCU not acquiring data 1 = DCU acquiring data
2	Ready	0 = Not ready to start acquisition 1 = Ready to start acquisition
3	Fault Status	0 = DCU Fault 1 = DCU OK
4	Adc Cal	0 = Not in calibration mode 1 = In ADC calibration mode

Each DCU sets its status field as follows:

Table 5: DCU Status Field Definition

State	*Status Word*	*Descriptor & Comments*
Power up	0x8	DCU Initialization State: On power up/reboot, each DCU completes its initialization and indicates an OK status, but not ready to acquire data.

Table 5: DCU Status Field Definition

State	*Status Word*	*Descriptor & Comments*
Stopped / Ready to Acquire	0xc	Ready to acquire: DCU has completed its initial configuration, as defined in the MAP/DIA, and is ready to start acquisition. This state is also reached when DCU is commanded from Acquisition state to Stop.
Acquisition	0xb	DCU is acquiring data normally, with no new data configurations pending.
Acquisition / New Config Pending	0xa	DCU is acquiring data and FrameBuilder has sent a new configuration request; DCU continues to acquire data while setting up for new data configuration.
Acqusition / Ready for New Config	0xf	DCU is acquiring data and is ready to acquire data using the new configuration on command from FrameBuilder.
ADC Calibration	0x100	ADCs are being calibrated.

In addition to the status information provided by each DCU, the DAQS controller also provides a status field, which needs to be checked by any other software which is attempting to use the MAP and DIA sections of reflected memory. Specifically, the 0 bit of the status word is set to ‘1’ whenever the MAP/DIA of the reflected memory is valid. In addition, processes should check the ‘run’ number in the IPC area to check when configurations have changed.

Table 6: DAQS Controller Status Field Definition

State	*Statuis Word*	*Descriptor & Comments*
Power up	0x8	Controller Initialization State: Controller is in power up initialization state. MAP/DIA are not valid for use.
Operational	0xb	Controller is acquiring data normally, with no new data configurations pending; MAP/DIA are valid.
Operational / New Config Pending	0xa	Controller acquiring normally, but is in process of reconfiguring the reflected memory. Other processes should consider the MAP/DIA invalid.

4.4. Memory Architecture Pointer (MAP) Area

The reflected memory MAP is a single structure of type memArch, as shown in Table 7, “struct memArch,” on page 12. This data structure provides information on the layout of the remaining

sections of the reflected memory. The MAP is filled in by the FrameBuilder each time a new data acquisition configuration is requested.

Table 7: struct memArch

Data class	*Variable Name*	*Descriptor & Comments*
char *	dataInfoPtr	Pointer to first dataInfo structure
int	dataCount	Total number of dataInfo structures (corresponds to total number of channels being acquired by the DAQS
char *	fastDataPtr	Pointer to first block of fast data
int	fastBlockOffset	Offset between fast data blocks
int	fastBlockCount	Total number of fast data blocks
char *	slowDataPtr	Pointer to first slowData structure
int	slowDataCount	Total number of slow data channels to acquire
char *	fftDataPtr	Pointer to first fftData structure
int	fftDataCount	Total number of fft data channels to acquire
char *	calDataInfoPtr	Pointer to calibration data definition (TBD)
char[32]	calDataFile	Name of file which contains calibration information
char *	calDataPtr	Pointer to start of calibration data block

4.5. Data Definition Area

The Data Definition Area describes the data within the Data area of the reflected memory. This data is described in a dataInfo structure, as defined in the following table.

Table 8: struct dataInfo

Data class	*Variable Name*	*Descriptor & Comments*
int	fbID	Number of FrameBuilder which is to store this channel; if Null, this data is acquired but not saved.
int	dcuID	Number of dcu which contains this channel
int	chNum	ADC channel number within dcu
char[32]	chName	Name of data channel

Table 8: struct dataInfo

Data class	*Variable Name*	*Descriptor & Comments*
char[32]	type	Data type (Short, Float, Double, etc.)
char[32]	dataBlock	Which memory block data is in: Fast/Slow
int	rate	Data rate for this channel
char *	dataPtr	Pointer to data location within first data block of reflected memory

4.6. Data Area

The data area is divided into four primary components:

- Slow Data: This contains information about the slow (1Hz) data channels along with the slow data.
- Fast Data Blocks (2): Location where the fast data resides.

4.6.1. Slow Data

The slow data area is only written to by the NDCU(s). It exists as a single 1Hz block of data. The structure is shown in Table 9, “struct slowData,” on page 13. The *engUnits* field is written to each time there is a new system configuration, with the data obtained from EPICS records.

Table 9: struct slowData

Data class	*Variable Name*	*Descriptor & Comments*
char[32]	engUnits	Engineering units of data
int	status	data status
float	data	data value

4.6.2. Fast Data

Fast data is stored in two data blocks, each data block containing 1/16 second of data, for a total of 1/8 second of data in memory at any given time. Data for each channel is stored contiguously as:

- status word
- Data values, the number of values dependent on the data acquisition rate for that channel (nElements = acqRate / 16)

Table 10: struct fastData

Data class	*Variable Name*	*Descriptor & Comments*
int	status	Data status; bit0 = valid/invalid, bit1 = in range/overrange
short	data	Data rate / 16 data points/block

5 LIBRARY CALLS

Several library calls are provided for use in other software to find data in the reflected memory and for notification that new data has been acquired. A brief description is given here, with further documentation to follow is the software comes to final design and implementation.

5.1. int daqnetChannelFind (char *name, struct dataInfo *data)

The calling process provides the name (*char * name*) of the adc to be located. This routine writes the reflected memory information on the requested channel in the dataInfo structure. The routine returns a ‘0’ if the channel is found or, if this routine cannot locate the requested data, it returns a ‘-1’.

5.2. int daqnetChannelList(struct dataInfo *list, int *nElements)

The routine is used to get a complete list of all data channels in the reflected memory. The routine sets the pointer to the first element in the list and nElements will contain the total number of data elements in the list. This routine will return a ‘-1’ if it fails.

5.3. int daqnetDataRdyFast(int *slowDataRdy)

When called, this routine will return when the next block of data has been received in the reflected memory (each 1/16 sec). The value returned will be the *offset* from the base address returned by *ChannelFind*, thereby pointing to the latest adc (fast) data block received. If the DAQS system has started a new run, indicating a new data configuration, this routine will return a -2, indicating to calling processes that they need to update their data pointer lists by recalling *channelFind*. A return value of -1 indicates a system fault. This routine will also write a value into the location pointed to by *slowDataRdy* (0 = no new data, 1 = new data ready), to indicate that new slow data and/or FFT data is available. This is done to allow a single call for processes which are looking for both 16Hz and 1Hz data.

5.4. int daqnetDataRdySlow()

This call will return a ‘0’ when new slow or FFT data is available. A return of ‘-1’ indicates a system fault and a return of ‘-2’ indicates that the reflected memory has been reconfigured and the calling process needs to relocate data with *channelFind*.