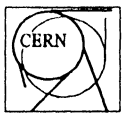
ALEPH 88-206
DATACQ 88-39
14 December 1988

D.R.Botterill, RAL



ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

Fastbus Database and Initialisation

Abstract

The Aleph Fastbus system will consist of over 100 crates containing over 1000 modules. A database has been created to contain the current status of the system, programs written to update the database and initialise Fastbus, and a subroutine package provided to allow access to the data.

Contents

1 Introduction

The Aleph Fastbus system will consist of over 100 crates containing over 1000 modules. A database has been created to contain the current status of the system, programs written to update the database and initialise Fastbus, and a subroutine package provided to allow access to the data.

For the databse to be useful, it should be updated whenever a Fastbus crate or module is added or removed from the Fastbus system. When an item is first added to the database, a name or number should be chosen that is meaningful to the users of that item. This will make subsequent modifications much easier.

The database is detector-independent, and detector-specific configuration tables should be derived from the Fastbus database and converted to a form most useful to the sub-detector.

Fastbus can be dynamically re-configured to allow or dis-allow modules in one segment to access modules in another segment, separately for broadcast and for normal Fastbus actions, to control the passing of the interrupt line (SR) up the tree and to re-assign arbitration levels. However the configuration must always satisfy certain rules, for example two modules in the same segment cannot use the same arbitration level.

Although in the simplest forms of partitioning, the fastbus tree can be split into sub-trees, in the more general forms, partitions will be sharing segments,SIs, and SRHs. This can be best handled by collecting the information in one place and having one routine merge the requirements of all the partitions and re-configure Fastbus appropiately.

2 Program containing menu access to the database, and to initialise Fastbus

The program can be run stand-alone by

$ RUN A_FBI$DIR:FBINIT

if one has SYSGBL privilege; it is also in the scheduler tables on AL0VOL and AL0VHC as FBINIT.

The top menu allows one to select the set of database tables to be inspected or modified, or to select the actions to perform on Fastbus.

The database table menus are in a common style with some variants. The top section allows one to select a subset of items to list, to add a new item and sometimes other actions. The bottom section is a pseudo parameter page where the parameters of one item can be changed. Note that changes are only effective if the "Accept" option is selected.

The one item to be changed is selected either by sufficiently specifiying the list criteria, or by further selecting one item in another menu which displays the list of items satisfying the criteria.

2.1 The information that can be changed by menu

The information that can be changed by menus includes:

1) Specifications of types of Fastbus devices, which contains the name of the type,number of slots it occupies in a crate; and for each port (Fastbus addressable) the Fastbus ID, whether it connects to a crate or cable, and for crate ports, the offset of the addressable slot from the right-most slot occupied.

2) The current layout of the Fastbus system:

- segments: whether a crate or cable or crate cluster and its GPs.
- devices: device type, Fastbus addresses (GPs and GAs), absent or not. Note that the connectivity of crates and cables is implied by the presence of SIs.
- VAX interface: the relation of EVIs to VAXs

3) The use that needs to be made of the layout. In each case, the end points of the transaction are specified and not the intervening SIs. Although menus are provided for these tables, it is expected that the more normal mode of operation will be that these tables be manipulated by the partition controllers to add and delete entries for each of their needs.

- broadcasts.
- SR source and handlers.
- routes for normal Fastbus operations. These can be defined two ways:

type=multiple: from start segment to all segments lower down the tree; a "multiple" route from the top segment in the tree allows every segment to address everything below it.

type=simple: from start to end segment. only useful for routes up the tree providing a "multiple" route exists.

Data should be entered in the following order:

1 Any new device types

2 Segments

3 Devices - at least SIs, AEBs with EVIs.

4 VAX interfaces

5 Useage

6 The "search" facility described below can be used for other modules.

The program makes some checks on the consistency of the data before modifying the database, for example GPs and Fastbus addresses must be unique and crate slots cannot be occupied twice. Devices marked as "absent" are ignored in these checks. More subtle inconsistencies may only become apparent when the route tables are computed.

2.2 The Fastbus actions

Menus exist to cause the following actions on Fastbus:

- Initialising Fastbus, either totally or starting at one segment and going further down the tree
- Reading the available Fastbus registers to check if the SIs. etc still contain the values loaded in them.
- Reading all Fastbus addresses to look for modules that are not in the database. This can be a convenient way to enter most modules in the system, although it cannot handle correctly modules that have two Fastbus addresses such as AEBs,TPPs etc.

Before any of these actions, the program checks if the database has been changed by any menu. In this case the tree structure and routing tables are recalculated and stored in the global section. Otherwise the existing data is used.

3 Program access to the data in the database

The data is stored in a disk based global section which should be mapped in a program's initialisation phase by:

iret=CREATE_FB_GLOBAL(' ')

to map to the global section with read access,or

iret=CREATE_FB_GLOBAL('WRITE')

to map with write access.

The following must be included in the link commands:

A_GBL$DIR:CREATE_GLOBAL,GBLSEC_OPEN,GBL_ERRORS

A_FBDB$DIR:ALL_FB_CLUS/OPT

A_FBI$DIR:FBDB_ERRORS,FBINIT/LIB

The data exists in Fortran Records defined by Structure statements. The first column is the Adamo ID, which is zero for rows not is use and equal to the row number for rows in use.

Some information is available by calling subroutines described below, otherwise the database can be searched by the routines that return a row number and then the information can be accessed in the Fortran structure. Most tables contain columns which are pointers (row numbers) in other tables, thus it should be very rare that the entire database need to searched.

Each table is defined by a separate include file in A_FBDB$SRC:table.INC and every definition can be included by the include file A_FBDB$SRC:ALL_FB.INC These include files contain a comment for every table and every column.

3.1 Introduction to the more important tables and links

"Ports" are the parts of a device that are addressable from Fastbus or which act as a master at a Fastbus address. They are held in table FBPORT whose columnns include:

- PNAME - the name of the port. Usually created from the name of the device.
- PRIMGADD - the Fastbus address
- ALREGVALUE - its arbitration level (if a master)
- ID_PTYPE - pointer to its port type,
- ID_DEVICE - pointer to its device and

- ID_SEGC - pointer to its segcomp (see below).

"Devices" are Fastbus modules and may have more than one port (eg AEBs,TPPs). They occupy slots in a crate. For the purposes of the database, the collection of one AEB CPU, several AEB memories and several EVIs connected by a front panel bus are considered to be one "device". Devices are held in table FBDEVICE whose columnns include:

- DNAME - the name of the device.
- DNAME - the name of the device.
- STARTSLOTNO - the slot number in a crate
- ID_DTYPE - pointers to the device type,
- FIRST_PORT - a pointer to the first port on the device. The other ports are chained together through the NEXT_ON_DEVICE column of the port table.
- IGNOREFLAG - a flag (logical) to indicate that the device is temporarily absent.

Similarly the Fastbus specifications are split between tables DEVICETYPE and PORTTYPE. The port type table contains the Fastbus ID and whether the port is a crate or cable port.

"Segcomps" are crates or cables with a unique GP , and "segments" consist of one "segcomp" or in the case of a crate cluster, a set of "segcomp"s. This reflects the hard-wired connection in a cluster.

Segments are held in table FBSEGMENT whose columnns include:

- SNAME - the name of the segment.
- ID_PORT_BC - pointer to the SI that leads up the tree; more exactly to the port of the SI on the next higher segment.
- BASEGP - the GP to be loaded in the Anciliary logic.
- ID_SEGC - a pointer to the first or only segcomp of this segment.

Segcomps are held in table SEGCOMP whose columnns include:

- GP - the GP of the segcomp (=BASEGP of the segment except for clusters)
- NEXT_ON_SEG - a pointer to the next segcomp of the segment, or zero.
- FIRST_PORT - a pointer to the first port on the segcomp. The other ports are chained together through the NEXT_ON_SEGC column of the port table.

The table GPALLOC is indexed by the GP number and contains ID_SEGC, a pointer to the "segcomp" of this GP. Using the chain of the "ports" on the "segcomp", the device corresponding to a Fastbus address can be found quickly.

4 Subroutines

This section describes the programming interface to the routines. It uses the same format as the VMS System Services Reference Manual.

fbdb_find_device

Find device of given name

FORMAT

fbdb_find_device *dev_name,id_dev*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

dev_name

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of device required

id_dev

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of device in table FBDEVICE; 0 if not found

fbdb_find_port

Find port of given name

FORMAT

fbdb_find_port *port_name,id_port*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

port_name

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of port required

id_port

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of port in table FBPORT; 0 if not found

fbdb_find_segment

Find segment of given name

FORMAT

fbdb_find_segment *seg_name,id_seg*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

seg_name

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of segment required

id_seg

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of segment in table FBSEGMENT; 0 if not found

fbdb_get_pa_of_port

Get primary fastbus address of port of given name

FORMAT

fbdb_get_pa_of_port *pname,primad*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

pname

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of port required

primad

VMS Usage: **longword_unsigned**
type: **integer**
access: **write only**
mechanism: **by reference**
Fastbus primary address; 0 if not found

fbdb_find_port_of_pa

Find port of given Fastbus primary address

FORMAT

fbdb_find_port_of_pa *primad,id_port*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

primad

VMS Usage: **longword_unsigned**
type: **integer**
access: **read only**
mechanism: **by reference**
Fastbus primary address

id_port

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of port in table FBPORT; 0 if not found

fbdb_find_devicetype

Find device type of given name

FORMAT

fbdb_find_devicetype *dev_type_name,id_dvt*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

dev_type_name

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of device type required

id_dvt

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of device type in table DEVICETYPE; 0 if not found

fbdb_find_porttype

Find port type of given name

FORMAT

fbdb_find_porttype *ptname,id_ptype*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

ptname

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of port type required

id_ptype

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of port type in table PORTTYPE; 0 if not found

fbdb_find_segtype

Find segment type of given name

FORMAT

fbdb_find_segtype *stname,id_stype*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

stname

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
Name of segment type required

id_stype

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of segment type in table SEGTYPE; 0 if not found

fbdb_find_vaxintf

Find host interface of given name

FORMAT

fbdb_find_vaxintf *host_name,id_proc*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

Longword condition value. Condition values that can be returned by this routine are listed under "RETURN VALUES".

ARGUMENTS

host_name

VMS Usage: **character string**
type: **string**
access: **read only**
mechanism: **by descriptor**
VAX DECNET name; blank for any interface

id_proc

VMS Usage: **longword_signed**
type: **integer**
access: **write only**
mechanism: **by reference**
Row number of VAX in table PROCINTF; 0 if not found

5 Error codes

RETURN VALUES

FBDB_OK	Indicates successful completion
FBDB_NOTFOUND	Indicates item was not found