VXI Basic Data Base Structure

Christoph Ender

1-Sep-1990 Draft 0.2

This Document tries to explain why a Data Base is necessary for the VXI Resource Manager. Furthermore a first attempt to define a basic Data Base for the VXI Resource Manager is done. The term Data Base used in this document defines the data structures and the relationship between the data structures. It has nothing to do with the event storage or event format definition, which some people have in mine if they call something a data base.

1 Introduction

Following the VXIbus specification one finds in part C of them a detailed description what to do during startup of a VXI system. One has to scan the geographical and logic address space to find SC (static configured) VXI-devices and DC (dynamic configurable). The information found in the VXI configuration registers has to be stored in a data structure owned by the VXI Resource Manager to allow them to configure the system *the DC devices* (refer to part F of the VXI specifications) and to initialize the modules found in the system.

For the EUROGAM project as well as for the other projects in Europe (i.e. EUROBALL, DEMON, ICAR, SUSAN)[1] a flexible solution is required, to allow upgrades and changes in the VXI based electronics. To reach this requested flexibility the most VXI devices will be dynamic configurable. This VXIbus modules has to initialized in their device dependent part according the experiment in which they are incorporated.

Therefore the VXI Data Base has to describe:

1. what's inside the VXI crate, which type of VXI devices, where it is located
2. how to initialize it
3. the result of the initialization
   (e.q. logical address, VMEbus address, status of the device ...)
4. Module constants to allow proper experiment initialization
5. Channel *physics channel* initialization to a basic operation which allows proper testing, and a save state between different modules in the crate.

This Informations are necessary on a per crate basis. **Note:** that the Data Base has to store all information which are stored by the VXI devices configuration registers. Due the fact that a register can have different meanings in read and write case. Some of them carries three or or four different meanings, just depending on the current content of them *an example is the message protocol register.*

---

[1] They might use the same general design of the data acquisition structure.

However in most applications we have in mind the systems consist out of several VXI crates and a few other VME crates. Therefore we need in addition to the crate based Data Base a more general Data Base somewhere in the system. This central Data Base contains information about

1. How many VXIbus crates,
2. where they are located,
3. how to reach them (Network address, ...),
4. what is inside the crate (partial copy of VXIcrate Data Base ...)
5. state of the crate
6. other hardware (CAMAC, VMEbus, TapeServer, ...)
7. experiment specific Data Base contains actual settings
8. read out relevant informations
   mapping of experiment channels to the actual hardware and the *Event Format Definition.*
9. logging and calibration Data Base

Most points of this overview have to deal the the overall design of the EUROBALL–Data Base and will be not discussed within this document. In the following only the VXI Data Base relevant parts and a mapping structure from a more general description to the VXI hardware is discussed.

As a start point in the design the Entity-Relationship Model is choosen, combined with modern SASD techniques (Data Flow Analysis, Performance Analysis). The mapping of the results into a Data Base will be done in a model as simple as possible, due to the requests from realtime constrains.

NOTE: *There will be changes in the mapping of the* Data Base*–design into our simple* Data Base*–model (Tabular form together with linked lists) during the development of the system and the forthcoming discussion.*

2 Overview

In this part a detailed overview about the relationship between the different parts of the Data Base is given. In figure ?? the different required components are shown.

In total we will have six tables. The VXIslot and the VXIdevice tables are required to allow a mapping to the VXIconfig table. One can identify these as index tables to speed up the access to the VXIconfig table in the case of a geographical (via VXIslot) or logical (via VXIdevice) addressing of a VXI module[2]. In addition both index tables will have some cross reference information. *In principle this is in some way a redundant information, specially if we use it in the central* Data Base. *In the case of the use of a full Relational Data Base System (RDBS) the redundant information can be omitted there. However the importance*

---

[2] Remark: A VXI module in a single slot can house several logical VXIdevices

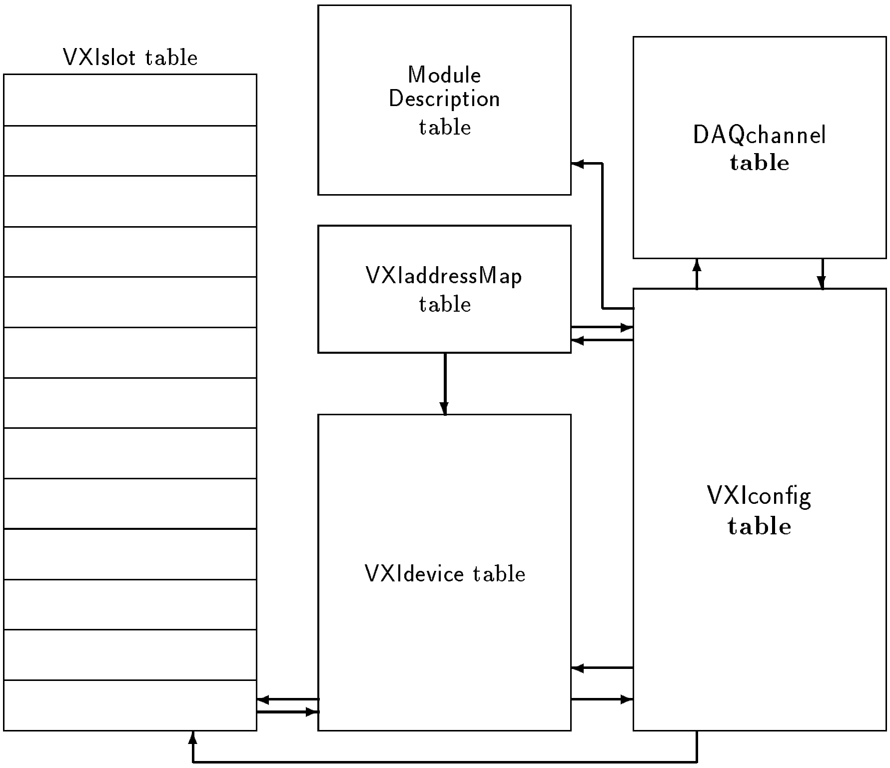

Figure 1: Overview and relationship between the different components of the VXI Data Base
.

of these cross links will be arise in the case of error handling during the normal operation where a fast response to certain conditions is necessary to ensure save operation of the Data Acquisition System.

In the VXIconfig table there will be stored the complete information over the status of the related VXIdevice, it's type and the associated link to the ModuleDescription table, and a link to the DAQchannel – table. Furthermore back links to the VXIslot and VXIdevice tables will be used. Another required link points to the VXIaddressMap table, which is used to construct the VXIbus address schema, as pointed out in the VXI specifications.

The ModuleDescription– table which will be discussed in detail in section ??. It contains a full description of the module and the description what to do during the different phases of initialization. Furthermore the basic initial data, test and diagnostics information is stored. For each Module type used only one entry in that table has to be used. The information is stored there in a position or logical/real address independent way, for use by any module under inspection/ access/ test/ diagnostic.

The just discussed organisation, which defines the relations between different entities, has the benefit that informations – out of the ModuleDescription, VXIconfig, VXIdevice, VXIslot, VXIaddressMap, and DAQchannel – has to stored only once. This speeds up also the access to the relevant information, due the fact that for a certain task only a specific set of information is necessary, which will be stored in one of the tables (e.q. Initialisation).

A critical part in the design of this structures is the detailed organisation which is very closely related to the VXI RM software design. The organisation of the structures and will be discussed in the appendix to this document.

3 Module Description Data Base

Module definition, what's in what's required should be in a form of a text file available, from which other programs can extract informations required for them. In addition Interpreters will be necessary to convert this information into a binary form. In the following a prototype of the a possible form is shown:

```
Module name;
  Description
    Type ..... , ......;
    Manufacturer .....;
    Channels ..... , .....;
    Hardware Description file ..... ;
    Software Description file ..... ;
    Submodules ..... ;
    BusSystem .....;
    { BusSystem dependent part (VXI|Fastbus|CAMAC|VME|.....)}
      Capability ..... , ..... ;
      VXIdevices ..... ;
      ....... ;
    EndDescription;
  Initial Phase n;
    DBcreate name, type , size, substructure , .....
      Record description in SQL or similar syntax;
      EndDBcreate
    Read address , type of access, expected data value [= DBentry.field] ;
    Write address , type of access, data value ;
    Test .... ;
    Sequence ...
      End Sequence ;
    Store data in DBentry.field;
    CheckForSubmodule address, name of expected Submodule, action();
    GenerateAddress ;
    ExecuteProgram filename ;
    EndInitial;
  Test 1
   Read, Write, Test, Sequence, ...
   EndTest 1;
  Test 2
   Read, Write, Test, Sequence, ...
   EndTest 2;
  Diagnostic
    Type ... Phase .... File ... ;
    required Diag ... ;
    EndDiagnostic;
 EndModule name ;
```

Each module requires first a description to define what type it is, what bus system it supports[3], the capability of the module and the type of submodules if there are any. The later has to be defined in a similar form. An example is shown below. Note that the syntax will be the same with one exception: the submodule keyword is not allowed.

```
SubModule name;
  Description
    Type ..... , ......;
    BusSystem .....;
    Manufacturer .....;
    Channels ..... , .....;
    Capability ..... , ..... ;
    Hardware Description file ..... ;
    Software Description file ..... ;
    { BusSystem dependent part (VXI|Fastbus|CAMAC|VME|Slave,FERA,VSB.....)}
     VXIdevices ..... ;
   EndDescription;
  Initial Phase n;
    Read address , expected Data ;
    Write .... ;
    Test .... ;
    Sequence ... ;
    End Sequence ;
    DBcreate name, type , size, substructure , .....
      Record description in SQL or similar syntax;
      EndDBcreate;
    EndInitial;
  Test 1
    Read, Write, Test, Sequence, ...
    EndTest 1;
  Test 2
    Read, Write, Test, Sequence, ...
    EndTest 2;
  Diagnostic
    Type ... Phase .... File ... ;
    required Diag ... ;
    EndDiagnostic;
  EndSubModule name ;
```

The further characterization of the module has to specify informations about the necessary module initialisation, test, and diagnosis. For this purpose this Module definition file contains additional subfields:

Initial: what to do during the Module initialisation

Test: declare Test sequences

---

[3] That's the main bus, the module itself may have several private busses (normally CPU's) where additional submodules will be connected.

Diagnostics: declare available Diagnostic

To allow a proper operation – during the several startup (initial) phases: *primary initial, test, autoconfiguration, diagnostic (extended tests), channel initial, channel tests, initial for normal operation (defaults), initial for experiment operation* and the during normal operation as well as during user requested tests and diagnostics – for each of the actions a phase number has to be specified. In this proposal phases 0 ... 10 should be assigned for execution during the startup of the VXI–System[4]. Each phase is finished completely before the next is allowed to be started: i.e. Initial Phase 0, Test Phase 0, Diagnostic Phase 0, Initial Phase 1 ,.... and so on. Phase numbers larger than 10 can be called by the user during data acquisition and phase number larger 100 specifies initial, tests and diagnostics sequences for standalone test, calibration and extended diagnostics, which can be requested only in test mode of the crate (locally via Console terminal or a connected workstation).

In the following subsection the detailed syntax will be defined. Note that there will be different representations of the Module Data base in the different parts of the System. In the VXI system only a subset of the Module description will be loaded, whereas in the central Data base all information is available. To extract the informations and to bring it in a useful form (binary) a set of programs are required. They should also be able to create the required include files, the data definition files, and an insert into the general data base after some syntax checks and ordering of the information found in the user supplied Module Definition file. There will be also comments allowed, specify them in the same syntax like "C" (i.e. `/* ... */`).

In the following subsections items set in typeface are keywords, text appear if *typeface* signal user specified input/variables/values.

3.1 Module Description

The module description is the most important part of the definition. It is enclosed in the keywords `Description` and `EndDescription;` . Inside this pair of keywords a fixed set of keywords and a bus type dependent keyword set is allowed.

<Type =*name*[,ID=*hhhh*][,OPT=*further informations*]> Type of module out of a fixed identifier set. ADC, TDC, Memory, Message, VXIDC, VXISC, VXIREG, VXIDAQ, CPU, VXIRM Other will be defined if it will be necessary.

< Manufacturer = *name*[,ID=*hhhh*];> specify the Manufacturer name of the module and as a optional argument the manufacture ID. The ID might be not available for all Bus Systems, for the VXI it is a must.

< HardwareDescription *filename*;> Define the filename/path in which the description, including the schematics will be found.

< SoftwareDescription *filename*;> the same for the software description.

< SubModules=*list of submodule names*;> Several Modules will consist out of several submodules. Some of the Modules will have one type others will hold different submodules. (One example will be the Flash ADC/Logic Analyser)

---

[4] This can be generalized to any bus system. In the definition it is already foreseen!

< BusSystem = *keyword* > Depending on the Bus System type there will be a special syntax.

VXI	defines the module to be a VXI device which fulfills the VXI specification in all points.
Slave	defines a module which have only two access registers one to set up the slave internal address and a data access register (i.e. FALA Submodule for the VXI-RM). Such a module can be a submodule of another module only.
VME	a normal VME device
VSB	a device which is connected to a VMEbus CPU by the VSB bus.
CAMAC	not defined up to now
FERA	A Fast Encoded Read device. *syntax not specified*
FASTBUS	*not defined up to now*
SCSI	*not defined up to now*
Ethernet	*not defined up to now*
User	*not defined up to now*

Depending on the specified bus type there are different keywords allowed. Each keyword have as the prefix the bus type to allow an easy implementation of the syntax scanner. In the following subsections we distinguish between the different bus types, because a keyword may have different meanings by the use of different bus types. *Only the keywords of full specified bus types are described.*

3.1.1 VXI

Define here all keywords allowed for VXI devices. Note all keywords have an additional prefix *VXI_* which is not shown here.

Capability = *A16 A24 A32 A64 ; D08 D16 D32 D64* supported access type to the module.

ManId = hhh; Manufactur Id

DevID = hhh; Device Id

AddressMask = hhhhhhhh; ¡text¿

AddressSize = hhhhhhhh; ¡text¿

BaseAddress = {fixed=aaaaa — dynamic }; define the base address for the VME bus address map. Normal it should be dynamic. Only for test and devices under development fixed addresses are allowed.

VXIdevtype = {Register, Memory, Message, Extended }, {DC, SC }; define the type of module and if that is a dynamic configurable or static configurable. For the device all required Registers are defined automatically, the optional ones has to be defined via the following keyword.

ConfReg = (add,),*(List of regs)*; Configuration registers which are defined in addition to the device type ones. Each Registers definition requires first the hexadecimal address, then the mnemonic for symbolic access together with the access mode (read only, write only, read write).

ReadConfRegMask = (*list*); list of registers which allows read access. It is a 32bit wide mask, for each register one bit.

WriteConfRegMask = (*list*); Register mask for write access.

SlavePort = (*AddressReg*, *DataReg*) Define Register pair to access a slave device.

Interrupts = (*Mnemonic*, *Vector*, *VectorSize*, *Level*), ... Specify the possible interrupts.

3.1.2 Slave

The Slave device is a device which has a register port. There is a 16bit address and a 16 bit data bus.

BaseAddress = nnnn, nnnn first address within the slave or a list of possible base addresses.

AddressMask address mask defined by 2 $(16 - n)$

AddressSize logical addressing space highest address minus base address.

Reg = (address, mnemonic, (R0,WO,RW,RC=cccc)), ... list of registers (16bit) defined by there address, mnemonic, and the access mode.

Block = (mnemonics, address, size, access), ... list of memory blocks.

3.1.3 VME

Similarly to the VXI devices the VMEbus devices have to be specified.

Capability= has to be specified

AddressMask ¡text¿

AddressSize ¡text¿

3.2 Module Initial

Read *address* [,*expected data*] = { *variable*— DBentry.field } ,

error = *(”output string”,variables,* { *abort*, *continue* } read a defined register and store content in a local variable[5] or in a database entry. Furthermore the result of the tread can be checked against an expected data value. If that is not equivalent an error is signaled, and written to the standard console device. Also exceptions like bus trap errors are signaled.

Write ¡text¿

Test ¡text¿

---

[5] allocated from a local private stack

Store	¡text¿
CheckForSubmodule	¡text¿
DBcreate	¡text¿
GenerateAddress	¡text¿
load program	¡text¿
execute program	

3.3 Module Test

Read address;	¡text¿
Write ;	¡text¿
Check ;	¡text¿
Store ;	¡text¿
LoadFile ;	¡text¿
Error ;	¡text¿

Execute program name;

3.4 Module Diagnostics

Type name Phase nnn File filename ¡text¿

Required Diagnostic list of

3.5 Required Programs

For proper operation and handling of the Module description there is a handling program necessary which allows the syntax check, a generation of a description table, and an "C" include file to generate a memory resistant version of the table. Furthermore it will be necessary to generate a TeX formatted file for documentation.

In the future there will be also the need to have a Data Base version of the module description available, which allows the generation of the Data Base data for the central Data Base–Server. From that a slave can then retrieve such informations not found in the memory base tables.

4 VXI Configuration Data Base

The configuration of the VXI required data base consists out of the following Data Base tables:

VXIslot	which contains informations about the type of the module, the VXI device number, how many VXI devices are in the module located, and a pointer to the VXIConfig table.

VXIdevice	Define the mapping table between the VXIdevice number, the module and the config table. This table has 256 entries with the following structure:
VXIaddressMap	Defines the Address mapping data structures which are required to construct the A24 and A32 address maps.
VXIconfig	This is the real configuration data table, containing all informations required to characterize a VXI device. This table is generated during the initialization of the VXI system.

4.1 VXIslot table

which contains informations about the type of the module, the VXI device number, how many VXI devices are in the module located, and a pointer to the VXIConfig table.

Mtype	Module Type
ID	Module ID (manufacturer id and module id)
NumVXdev	Number of VXI devices which are defined by this module
BasVXdev	VXI device number of the first VXI device of this module
pConfig	pointer to the first Configuration entry in the VXIConfig table

4.2 VXIdevice table

Define the mapping table between the VXIdevice number, the module and the config table. This table has 256 entries with the following structure:

Module	Module/slot number
SubModule	Sub Module/slot number (number of VXI device in this module)
pConfig	pointer to configuration entry

4.3 VXIaddressMap table

Defines the Address mapping data structures which are required to construct the A24 and A32 address maps.

next	pointer to the next AddrMap entry
pConfig	pointer to configuration entry
Address	access address
size	size in bytes
type	type of access a24/a32
m	copy of required memory size

4.4 VXIconfig table

This is the read configuration entry.

It contains all VXI configuration registers in each meaning. In addition a bit map with registers which are readonly, write - only, and not accessible. Furthermore it contains also some status information if this device fails.

The organisation of the is in form of a linked list. Indices pointing back to the Module mapping table.

prev	pointer to next and to previous entry in the Configuration table
user	user extension part
VXIdeviceNumber	logical address
VXIslotNumber	geographic address
AddressDesc	Pointer to associated Address map entry
ModuleDesc	pointer to the Module description
DAQchannel	pointer to DAQchannel
ReadAddressMask	Bit mask for all of the 32 Config registers, specifies which register can be read out without a bus error fault, and is defined as a readable register in the module table. *this entry is a work around to speed up access to this table.*
WriteAddressMask	Bits mask for writeable Configuration registers.
InvalidAddressMask	This registers can be not accessed (Bus Error occurs during access.)
RD	Readable configuration registers After a read of such a register it's content is written into this field.
WRT	Writeable configuration registers After a successful write the content has to be written into this field.

5 VXI Data Acquisition Data base

This part will contain all Information which are related to the data acquisition system. The information which has to be in that structure has to be:

1. Channel parameters, like channel numbers, ADC identifiers, parameters for experiment specific setting of ADC's, reg's, ...
2. Status Informations
3. Spectra Definitions
4. Rates for monitoring
5. Which Module, SubModule, VXIDevice is related to that channel. VMEbus address for access *Or more general expressed: some back link Informations are required for the easy mapping and debugging of the whole system. See also figure* ??

From the point of the VXI related Data Base it is not necessary to know what's really inside these records, but the general structure has to be created during the startup. Furthermore the default informations/parameters found in the Module data base *required for initial and proper operation during startup tests* have to be inserted as the current values into the channel Data Base. *More discussion required on that point!*

Up to now only the point of view from the VXI system is enlightened. For further discussion also the point of view from the global system or more specific spoken from the data acquisition application program (i.e. Rate Monitor, Calibration programs, ...) has to be discussed in detail.

A Implementation Proposal

To implement this database the structures has to been mapped to a sufficient form to store it memory and into data files. The mapping into data file is the preferred technique for the central Data Base which is not in the scope of this document. That task has to be solved by David Brightly.

In the following the mapping of the data structures into a memory based form is described, which is one of the very important and critical point of the design for the basic software structures for the VXI based software. (See also the **VXI Resource-Manager Basic Software** document.)

A.1 General

A data base can be stored quite easily in tabular form, which is the normal representation. All entries represent one and only one entity with certain attributes within the data base. This entity is called **record** in the further discussion. Each record contains the whole information required to describe the entity in a unique way.

There exist two organisation forms to store the records:

a) a fixed record structure with fixed length records

b) a variable record structure with variable length records

Both forms are widely used by data base programs.

In our case both forms will be used. The fixed organisation form is adequate to organize the index and mapping tables (VXIslot, VXIdevice, VXIaddressMap). These tables will contain only pointers, a few data values, and indexes.

However for the other Data Base tables (VXIconfig, ModuleDescription, DAQchannel) a variable structure is more useful due to the fact that the number of the required information for each record varies from entry to entry. Specially in the case of the ModuleDescription, due the fact that each module is different from the others and therefor other initialization, test, and diagnostics has to be executed. If such a structure is mapped into a fixed format one have to foreseen all possibilities which may occur during the live of such a data base. This allows no flexibility and absolutely no extension during the live. If that is necessary the whole data base has to be restructured, data base programs has to be modified or rewritten. *In the* EUROGAM *project with our time constrains the modifications will be a problem if we have to change then all our programs which requires access to this data structures.*

To overcome this problem the first point is to organize the records into a regular structure which will be used by all tables, including the fixed tables. This will be the record header with the following data entries

length :	total length of the record in bytes
ID :	Identifier which record type. *Equivalent to the data base table*
next :	pointer to next record (in the case of a variable heap organisation of the data base)
previous :	backward pointer

status : status word for use by record type dependent software. The highest bit will be used as a lock bit to be able to synchronize the access in a multi task environment.

control : Control word for use by record type dependent software

N_entries : Number of variable length items within this record

first item block :

second item block :

The item blocks are not part of the record header. The two pointers are required for the organisation of data base structures which will be generated on the fly which is the case for the VXIconfig and the DAQchannel table. These dynamic lists may change quite frequently. Furthermore that allows changes in the table structure on the fly by application programs, for example if one entry has to be removed from the channel table due to a malfunction.

The item blocks consists out of two parts, a fixed header and a variable section which is defined by the item block identifier. This identifier is defined for each information block as a unique number. The following fields defines the item header:

length : length of the item block, that's identical to the link to the next item block

ID : Item identifier.

variable part : the content of this field is defined by the identifier. This definition is available in the central data base, in include files which are generated out of the data base description, and in the application programs.

By using this structure a universal data base handling (the memory based part of it) can be used.

For each database an information block is required which describes the attributes of the data base, the name of it and contains the pointer to the records. Depending on the type of the data base this root block contains an index or a header for a linked list structure. In principle it should be possible to describe the following types of organisation of the data base :

1. fixed length record, indexed
   organised in form of an array
2. linked list
   This is a good organisation for small tables and in the case when pointers from an additional index data base will be used for access to such a table..
3. linked list with local index
   will be not used for VXI based data base
4. hash index linked
   will be not used by VXI data base

Beside that further information is necessary to define the minimal and maximal length of a record, the number of records in the data base, and a control and status word for synchronization of the access to the database and to disable concurrent manipulation of the data base structure. Another important attribute is the information whether the data base is dumped into a disk file or not, or if that is only a memory based data base. In the case of a disk file the location and the name has to be stored in this database header structure. Practical reasons make it necessary to add an entry which tells the software where the definition of the data base will be found.

In the following structure all fields in the header are described:

length : length of the data base definition structure

name : name of the data base

type : which data base type (organisation, disk/memory based)

index header :

linked list header :

Disk file : path to the disk data file

Structure Definition file : filename and path to the structure definition of this data base table

N_entry : Number of records in the table

Del_pointer : Pointer to deleted data base records (linked list structure)

status : Status word: in memory, disk updated, ...

access control : Synchronization locks (organized as semaphore) used for manipulation of pointers, indexes, dump, read back, clean up, and index calculation

created : date and time of creation

modified : date and time of last modification of a data value

last entry : date and time of the last insertion of a record

last access : date and time of last access

reads : number of record reads in the data base

writes : number of record writes in the data base

locks : number of record locks in the data base

It is also necessary to have a common data base directory, where pointers to all databases used will be found. The organisation is a simple index with pointer to the data base description block, where as already shown all necessary information will be found.

In the following some organisation details of the most important tables will be shown.

A.2 VXIconfig

The VXIconfig table contains all relevant information about all devices found in the VXI crate. It contains the links to all other tables. This information ha to be stored in the first item block.

LogDevAdd : logical device address

ControlStatus : Control and Status word

SlotNumber : Number of VXI slot in which the device will be found

DeviceNumber : VXI logical device number

ModuleDesc : pointer to the module description

AddressMap : pointer to the address map

Channel : pointer to the first DAQchannel entry

N_channel : number of channels

VXICread : copy of VXI configuration registers as been read

VXICwrite : copy of VXI configuration registers as been written

extentions : extensions to the VXIconfig which will be added during a later step (e.q. configuration constants).

All other information will be found in the next item blocks (see therefore section ??).

A.3 ModuleDescription

The ModuleDescription table contain the module description, initial, test, and diagnostic information. The table entry consists of an basic item block and several description blocks. The initial, test, and diagnostics descriptions are connected via linked lists to the basic item block as shown here. For each entry of the phase model there will be a own descriptor block (details will be found in ??, ??, ??).

name : name of module

Btype : bus type identifier

ModuleID : Module identification number

N_desc : number of module description entries

initial : pointer to initial block linked list

test : pointer to test block linked list

diag pointer to diagnostics linked list

The basic organisation of that table is the form of a linked list. It will consist out of a small set of records (< 30).

A.4 DAQchannel

B Structure and Item Code Definitions

B.1 Record header

Structure definition:

length :*unsigned short int*	length of the record in bytes
ID :*unsigned short int*	Identifier upper 8 bit defines the record type the lower 8 bit used for item and sub records
next :*unsigned int*	address pointer to next record a relative form used.
previous :*unsigned int*	address pointer to previous entry if a double linked list structure is used, or pointer to data base descriptor
control :*unsigned short int*	Control word for use by record type dependent software
status :*unsigned char*	status word for use by record type dependent software.
N_entries :*unsigned char (0 ... 255)*	Number of variable length items within this record

B.2 Item header

length : length of the item block, that's identical to the link to the next item block

ID : Item identifier.

variable part : the content of this field is defined by the identifier. This definition is available in the central data base, in include files which are generated out of the data base description, and in the application programs.

B.3 VXIslot

B.4 VXIdevice

B.5 VXIaddressMap

B.6 VXIconfig

B.7 ModuleDescription

B.7.1 Description

B.7.2 Initial

B.7.3 Test

B.7.4 Diagnostics

B.8 DAQchannel