VXI Resource-Manager Basic Software

Christoph Ender

3-Sep-1990 Draft 0.4

This version of document is a draft one. Changes will be made with the progress of discussion. It is for EUROGAM internal use only ! Do not distribute this draft document to any other organisation outside the EUROGAM collaboration.

Note: together with this version of the software the phase model of the VXI/DAQ tasks, processes is introduced. Together with that and other more refined definitions of the ModuleDescription data base structure it allows a very detailed and fine tuned control over each step of operation in the live (that means from power up to controlled power down, and during error recovery) of the software.

1 Introduction

This document specifies the basic software of the VXI-RM in terms of their function, structure, and data. It defines further the detailed interface between the RM and the user/application software including a network based RPC access.

The aim of this software is to write an operation system independent software which is executed in an early stage of the initialization of the VXI-RM processor. This has to be done on a crate basis. In a later stage the connection to the other parts of the system (central data base, data acquisition control, Event formatter, ...) will be done.

2 Overview

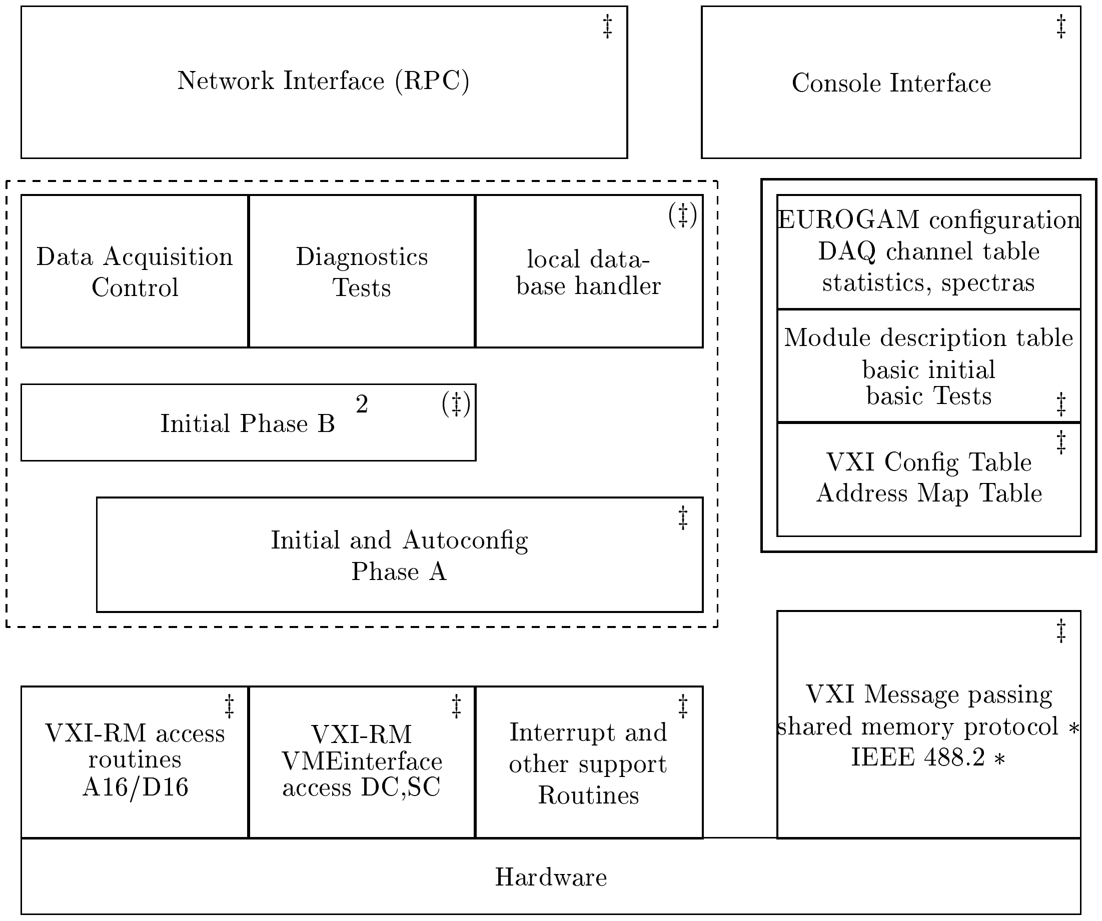
The overall structure of the software is shown in Fig. **??**. This picture gives an overview to the parts related to this document. In addition the relationship between the data base structures and the software is shown. The required data base tables in a brief overview are displayed within a double lined box, the main programs are within the dashed box.

To meet the VXI-specifications it is required to split the execution into several phases as shown in table **??**. More details how this will work are explained in section **??**

Access to the software from an external (other CPU, another crate) will be available via two ways:

1. Network Interface using RPC calls, they are ontop of the TCP/IP network system. That will be the normal way of operation for all network interaction, only a few requests will response to multicast operations.

2. Console Interface
   This part is required for a test interface, to display urgent error messages onto the console terminal, and to interactive use of a VXIcrate for local tests and interactive



Network Interface (RPC) ‡
Console Interface ‡
Data Acquisition Control
Diagnostics Tests
local data-base handler (‡)
Initial Phase B 2 (‡)
Initial and Autoconfig Phase A ‡
EUROGAM configuration DAQ channel table statistics, spectras
Module description table basic initial basic Tests ‡
VXI Config Table Address Map Table ‡
VXI Message passing shared memory protocol ∗ IEEE 488.2 ∗ ‡
VXI-RM access routines A16/D16 ‡
VXI-RM VMEinterface access DC,SC ‡
Interrupt and other support Routines ‡
Hardware

Figure 1: Software Overview. Parts marked with a ∗ will be not implemented during the first phase. Boxes marked with a ‡ are part of the VXI-RM software. Sometimes only a part of them is really related to the basic resource manager functions, another usage (not in EUROGAM) will need only a subset. The parts in the dashed box are the programs which are executed, all others are part of the support routines. The part with the double box contains the data base tables.

Table 1: Phases of operation

No.	Phase Model		Description
			Phase A
1.		ini	RM Software startup
.0	0		primary bootstrap of CPU
.a	1	ini_a	call of VXI-Initial Software
.b	2	ini_b	initial of internal data structures and variables
.c	3	ini_c	reset RM Registers by software
.d	4	ini_d	initial own VXI Registers
.e	5	ini_e	bring own registers in operation
.f	6	ini_f	check for possible salves and reset them
.g	7	ini_g	set end of primary initialisation
			Test Phase A
2.		stst	Self test
.a	8	stst_a	self test of VXI-Registers (read/write functionality)
.b	9	stst_b	test available slaves (FADC, P3)
			Phase A
3.	10	aconf	Auto configure system
4.		start	start up of the operating system
.			operating system is up, network is running
			Phase B
5.	11	saconf	Secondary Autoconfig and Initial
.	12	satst	Secondary tests and diagnostics
6.	15	netw	Start of Network part and connection to common data base
.	16	netwsynch	synchronize all crates
			Phase C
7.	20	oper	Normal operation

diagnostics. Unfortunately the implementation will be in part dependent on the operation system (VxWorks, OS9, ...). The planned software has to allow some primitive access to the VXI registers, and to the local/global data base.

3 RM-basic software

This will be some kind of interface routines to access the VXI-RM via its logical and geographic address space using A16/D16 access cycles. It is necessary to prepare the basic initialisation to set up the configuration tables which will be filled out in detail in a later step.

The basic software routines covers the following parts:

- basic access routines (read/write short word) with logical addressing using A16/D16 access
- geographical addressing using the MODID register
- logical addressing using A16/D16 address
- creation and access to the VXIslot, VXIdevice, and the VXIconfigtables
- creation and manipulation of Module Table
- creation and manipulation of the channel Table
- creation and manipulation of the address mapping table
- logical addressing using A16/D16 address
- access routines via A24 or A32

Beside this routines some additions for the error handling is necessary. The most important is the control of the access within the VXI crate during VXIbus/VMEbus transfers. They can be controlled for the right bus protocol using the CPU hardware. This hardware has to report the error to the access routines. *Implementation via the Bus Trap Error Routines.*

3.1 RM initial Software

This part is described in detail within the VXI spec's section C.4.1 .

The first step has to be the primary initial the RM itself and the initial of it's own logical address to 0 if the RM is itself a DC VXIbus device. With this initial a first test to check the address bus and the databus via the P1/P2 adapter card should be foreseen.

3.2 Slave Interface

Beside the test and initial of the VXI-RM parts the first test of the slave part and the required initialization has to be done.

The basic software will contain some routine to access the slave part. They include the access to registers and allow block transfer in 16bit or 8 bit mode.

At the moment three different slaves has to be initialized:

1. **P2 A+C row adapter card**

   More Information will be found in a document by R. Öhlschläger.

2. **P3 card**

   This card contains the basic registers, which deal with the trigger lines. Currently four registers are defined for that purpose. The initial has to make sure that the drivers are all in an inactive state. *A full specification will be available in late September. After that the software requirements can be specified.*

3. **FALA card** a combination of a flash ADC system and a small set of logic analyzer system.

   This card will have 32 registers. With the initial bit set in the control and status register of this device it will be inactive. The next step will be the setup to the default values to allow simple operation via the console interface. Details over that card will be found within the FALA specification.

4 Phase Model of the VXI Software

During definition of the software one reaches the point where several phases of initial and tests has to be controlled. The easiest way is to allow for each phase several actions like *initial, test, diagnostics* which are executed in that order. At the end of each phase the phase control software checks that all tasks which may be stared during the execution of the different action have finished and passes then the control to the next phase of operation.

With this general organisation one can organize the required actions of the VXI-RM startup into this phase model. Also the phases which will occur later can be described very easily in this picture. **The only thing which is necessary is a clear definition about the phases during the live of a processor / process / task, which has to be used by all software components involved in the EUROGAM all successor experiments**.

The table **??** might be a first definition of such a system.

Table 2: Phase model of the Data Acquisition Software live. This includes all phases which has to be synchronized.

Local startup phase	
0	primary boot
Global Synch phase	
20	wait for all crates available
40	Start Data Acquisition
Global Asynchronous Operation Phase	
100	General Consistency Check
Local Test and diagnostic	
200	Test all Modules

In this model phases 0 ... 20 are local, per CPU defined phases which can be executed from the system point of view asynchronously. After that all phases with number between 20 ... 99 requires synchronisation between the different parts of the data acquisition system.

5 Initial Phase (Auto Configure)

The initial phase takes place after the VXI-RM initial. It is called also Phase A and occupies in the Phase model step 0 ... 10.

1. Identify all VXIbus devices in the crate (system) and write there current data values into the VXI configuration database.

2. Configure all resources required for the proper VXI-RM operation (allocate memory, setup location monitor for message based operation)

3. start system self test and diagnostic sequence

4. construct A24 and A32 address maps. Therefore read address space requirements and configure table in a way that there is no overlapping area. For both parts a separate map is required. *The AM (access modifiers) ensures that there is no overlap even if the address may be the same ! Nevertheless the overlap of addresses should be avoided.* Load after construction of the addresses the address registers.

5. Configure commander/servant hierarchy
   At this time the primary selftest of the VXIdevice modules has to be ready for all devices. Then the software has to prepare the RM in such a way that a primitive message based protocol can be used. All associated message based devices has to be someone's servant, at least the RM has to be a servant.

   For the first phase of the EUROGAM project the VXI devices this will be implemented in the simplest possible way, a register based device. therefore only a very simple way of initialisation of the VXI-RM message protocol registers is required, There will be in the near future (beginning 1991) no need to build message based modules. To allow then the incorporation of other modules in a second phase of development of the RM software a full set of message handling routines will be provided together with an implementation of a 488.2 interface.

6. Initial normal operation
   Bring all message based devices in operation.

After that the VXI crate is in a normal operation mode. For initial of the VXIbus devices for use by EUROGAM further device dependent initialization is necessary. This will done after the operating system of the VXI-RM is up and running.

6 Secondary Auto Configure

After the OS and the network is up and running we have access to the central database. During this phase extended device tests and initial will take place. The information for this tests and initial will be found in part in the module description, which is necessary even in the primary initial phase to identify the modules itself. The other part of the information will be stored in the central database. Therefore the VXIcrate has to interact with the data base.

The connect between the two systems can be done via a broadcast request (**Broadcast-all-RPC**) to the data base. The broadcast is necessary due the fact that the RM do not know the network address of the data base server, on the other hand the data base server also might not know the network address of the VXI crate. After the acknowledge of the initial broadcast the normal interaction between the system will be done with RPC messages. If there is no acknowledge within a certain time the request is repeated for up to 100 times (5Min). If there is still no acknowledge after that time the VXIcrate will go into a local mode only.

To bring the VXIcrate up, even in the case that the data base server was not available within 5Min after network initial, it is allowed to access the VXIcrate by some kind of data acquisition software with a *claim crate* procedure. After that the secondary Autoconfigure phase may be requested via a RPC call from the data base server or by the data acquisition control.

After the connect between the VXI crate and the central data base they has to exchange the VXIcrate internal database. The information which have to be transported to the central database are:

1. how many modules, VXI RM internal configuration, network address
2. VXIdevice information, Module identifiers, logical and geographic address, device capabilities, address mapping (A24/A32 address and size),
3. status information over failed modules

With this information the VXI-RM can retrieve module dependent test information from the database and start the VXI device diagnostic tests (see a separate paper, which describes the module database and test specification software). After the tests are executed the results are written back to the data base.

If the device has passed this tests it has to be initialized with some values which brings the data acquisition modules in principle operation (default values).

After the last VXIdevice has passed this test and initial sequence the VXI-RM will forward a message to the central data base and/or data acquisition control that it has finished the local configure, test and initial phase.

The VXIcrate can be used after that in by the acquisition control to start in the VXIcrate another task which will do the load of the experiment specific values. Other tasks will be available for online diagnostics and tests.

All these tasks has to go through common routines which provide the VXIcrate local locking of modules (or part of modules). The necessary locking information will be located inside the VXIcrate configuration database.

7 Primitive Database

The VXI-RM has to setup for his own use a VXI system (as far as the RM can access VXI crates) data base. For diagnostic this database should contain a copy of the configuration registers. But not only that one which can be read, also the hidden registers (note that some registers carries several information read/write functions are different). Beside that information further areas for locking of a module. To allow a more general use of that database it has to be VXIdevice independent.

The device dependent information will be found in the Module Data base. This describes the modules itself and there requirements which will be not found in the VXI configuration registers.

Information found in the Module data base and the VXIconfiguration database allows the VXI-RM to construct a data acquisition data base which can be used by the data acquisition. This data base table will contain all device dependent status information, spectras,... .

7.1 VXI Configuration Data Base

This part contains the a copy of all VXI configuration registers. Dependent of the device type (Memory, message, register, extended) all registers are stored here. The access routines to the VXI configuration space have to store here the information, which then can be used for diagnostic and status informations.

The access to this data base should be done by a mapping table to get the right position within this table to store or to read out the information. Due to the fact that a read of the original VXI configuration registers does not give the right information back, bit manipulation operations need this copy of the data content of the hidden data register.

The VXI Configuration data base has to index paths, one via the Slot number and the second via the VXI logical device number (0 ... 254), the address 255 is reserved for dynamic configurable devices.

7.2 Module description data base

To identify and to describe a module some informations over that are required. This information will be found in this table. It will be include also some basic tests as well as the identification where to find further diagnostics and test programs.

The detailed form will be found in the data base specification.

8 Network Interface

The Network Interface is required to allow the Data Acquisition system to send commands, setup parameter, and control information to the VXI system. It acts then as in term of an data acquisition crate, and performs the so called crate functions. There the crate has to do functions like diagnostics, tests, and tuning of the data acquisition parameters itself like gains, offset, CFD parameters, ...

In principle all the data can be divided into several classes.

1. access to the VXI modules via the VMEbus address space or the VXIspecific geographic/logical addressing specs.
2. access to the VXI-RM specific slaves like the FADC, P3 sub card, and some of the special slave functions
3. access to the different data base tables, to load or retrieve values
4. access to the message based functions
5. access to the primitive functions and the control over the RM

Up to now only the external access to the crate is explained. The VXI-RM base software should have also the direct active access to the system. One point therefore is the communication with the Data base server to store there error– and other useful logging informations.

The other point is to request the specific functions in the case of initialisation, power error, or other important conditions which requires an immediate action.

As the basic organisation structure an RPC interface is foreseen. There all functions of the basic software can be connected to the RPC interface quite easily.

At the moment it is not clear for me, which type of RPC model the proposed RPC interface has to follow. Problems will be rising if we have several concurrent tasks, each with it's own connection to a RPC channel, which can produce under certain conditions an deadlock. Also it is not clear what will happen when several tasks (user application, data base server for periodic requests, data acquisition control) on different CPU's requests the same RPC channel on the same target CPU. Implements the SUN RPC an asynchronous or a synchronous operation model ???

9 Console Interface

This interface allows to access in a primitive way the VXI crate. It will be constructed around an interpreter. Beside the very primitive operations like read and write operations, also access to the VXI-RM databases are necessary.

to be completed

10 Module Dependent Basic Configuration

This part also called Autoconfiguration part two will be implemented in a type of data base scan, where the VXI-RM software will retrieve information from the database to execute then the program/code/actions found there. This way of operation allows to define an abstract interface without knowing what type of interfaces will be used in future.

It is foreseen that an unlimited number of configuration actions can be specified in the Module description data base (a text file).

A Call Interface

Here the calls to the basic routines are specified. The C specification syntax is used.

A.1 VXI access functions

VXi access functions are these functions which write, read, or manipulate the VXI configuration registers, including the message handling.

- basic access routines (read/write short word) with logical addressing using A16/D16 access
  - `status = VXIreadLogA16D16(LogicalAddress,Offset,&Data)`
  - `status = VXIwriteLogA16D16(LogicalAddress,Offset,Data)`

    `LogicalAddress` The VXI logical device number

    `Offset` Offset within VXI device configuration area

    `Data` 16bit data word to read or to write

`Status` The routines gives back a status words to signal that the data are transferred successful by the hardware (no Bus Error), that the LogicalAddress is within a valid range[1]. In addition the offset is also tested for a valid range $(0...63)_{10}$.

- geographical addressing using the MODID register
  - `status = VXIreadGeoA16D16(SlotNr,LogicalAddress,Offset,&Data)`
  - `status = VXIwriteGeoA16D16(SlotNr,LogicalAddress,Offset,Data)`

    Same as above

    `SlotNr` VXIcrate slot number to access Modules using the MODID line

A.2 VME access functions

- logical addressing using A16/D16 address
  - `status = VXIreadA16D16(LogicalAddress,&Data)`
  - `status = VXIwriteA16D16(LogicalAddress,Data)`
  - `status = VXIreadBlockA16D16(LogicalAddress,Number,&Data)`
  - `status = VXIwriteBlockA16D16(LogicalAddress,Number,&Data)`
- access routines via A24 or A32
  - `status = VXIreadA24D16(Address,&Data)`
  - `status = VXIwriteA24D16(Address,Data)`
  - `status = VXIreadBlockA24D16(Address,SizeOfBlock,&Data)`
  - `status = VXIwriteBlockA24D16(Address,SizeOfBlock,&Data)`
  - `status = VXIreadA32D16(Address,&Data)`
  - `status = VXIwriteA32D16(Address,Data)`
  - `status = VXIreadBlockA32D16(Address,SizeOfBlock,&Data)`
  - `status = VXIwriteBlockA32D16(Address,SizeOfBlock,&Data)`
  - `status = VXIreadA24D32(Address,&Data)`
  - `status = VXIwriteA24D32(Address,Data)`
  - `status = VXIreadBlockA24D32(Address,SizeOfBlock,&Data)`
  - `status = VXIwriteBlockA24D32(Address,SizeOfBlock,&Data)`
  - `status = VXIreadA32D32(Address,&Data)`
  - `status = VXIwriteA32D32(Address,Data)`
  - `status = VXIreadBlockA32D32(Address,SizeOfBlock,&Data)`
  - `status = VXIwriteBlockA32D32(Address,SizeOfBlock,&Data)`

[1]After the initial only access to autoconfigured VXIdevices is allowed.

A.3 Slave Functions

To access the VXI-RMslave interfaces six functions can provide a full functional programming interface.

- `err = slave_read(slave_addr,&data)`
- `err = slave_write(slave_addr,data)`
- `err = slave_read_block(slave_addr,N_datawords,&data)`
  16 Bit transfer
- `err = slave_read_block8(slave_addr,N_datawords,&data)` 8 bit data transfer
- `err = slave_write_block(slave_addr,N_datawords,&data)`
- `err = slave_write_block8(slave_addr,N_datawords,&data)`

A.4 Data Base functions

- creation of VXIdevice configuration table The access to the configuration Table can be done in two ways: first to use the geographical access to select a module, or by use the logical addressing by specification of the VXIbus device (0...254). The configuration table is constructed in a way that both allows the access to all data.
  - `status = VXIDBcreateConfigTable(TableAddress , SizeOfEntry)`
  - `status = VXIDBinsertConfigTable(SlotNumber , Offset , Data)`
  - `status = VXIDBgetConfigTableEntry(SlotNumber , &DBrecord)`
  - `status = VXIDBputConfigTableEntry(SlotNumber , &DBrecord)`
  - `status = VXIDBinsertDeviceTableEntry(LogicalAddress , Offset , &DBrecord)`
  - `status = VXIDBgetDeviceTableEntry(LogicalAddress , &DBrecord)`
  - `status = VXIDBputDeviceTableEntry(LogicalAddress , &DBrecord)`
- creation and manipulation of Module Table
  - `status = VXIDBcreateModuleTable(TableAddress , SizeOfEntry , NumberOfEntries)`
  - `status = VXIDBinsertModuleTable(ModuleID , TypeOfEntry , &MDBentry)`
  - `status = VXIDBfindInModuleTable(ModuleID , TypeOfEntry , &MDBrecord)`
  - `status = VXIDBgetModuleTable(ModuleID , TypeOfEntry , &MDBrecord)`
  - `status = VXIDBputModuleTable(ModuleID , TypeOfEntry , &MDBrecord)`
- creation and manipulation of the channel Table
  - `status = VXIDBcreateChannelTable(TableAddress , SizeOfEntry , NumberOfEntries)`
  - `status = VXIDBinsertChannelTable()`
  - `status = VXIDBfindChannelTable()`
  - `status = VXIDBgetChannelTable()`
  - `status = VXIDBputChannelTable()`

A.5 Autoconfigure Functions

- creation and manipulation of the address mapping table
  - `status = VXIDBcreateAddressMapTable(TableAddress)`
  - `status = VXIDBinsertAddressMapEntry(Type , VXIDeviceNumber , Size)`
  - `status = VXIDBconstructAddressMap(Type)`
  - `status = VXIDBgetAddressMapEntry(Type , VXIdeviceNumber , AddressOfset)`
- related to overall autoconfiguration
  - `status = VXI_AutoConfig_initial()`
  - `status = VXI_AutoConfig_PhaseA()`
  - `status = VXI_AutoConfig_PhaseB()`
- other support routines like Bus Trap Error, error display

A.6 Access of VXIsystem Functions

There will be two different ways of access: local and global. In this section the local access (also stand alone access) is addressed.

- Console Interface
- Reset functions

A.7 Network specific part

- access to other crates
- access to data base server
- access from other crates
- access from data base server
- access from application programs
- Reset system
- error logging

B Software Modules (Overview)

B.1 Definition Files

The definition files can be divided into several classes:

1. CPU and Operating System type specific: to hide system dependencies. This is not really necessary for EUROGAM because of the use of the MVM147 CPU under VxWorks. For the development other Systems will be used. There and even in a later phase of the project there might be changes, so we need some universal definition files to hide all dependencies.

   The files are:

- CPU_E5.H
- CPU_MVM147.H
- CPU_VAX.H
- OS_OS9.H
- OS_VMS.H
- OS_VXWORKS.H

2. Universal VXI definitions will be also available
   - VXI_CONF_REG.H
3. The Resource Manager Data Registers, the flash ADC part, and P2/P3 card
   - VXI_EGRM.H
   - VXI_FALA.H
   - VXI_P3.H
4. Data Base part
   - VXI_DB_CONF.H
   - VXI_DB_DAQ_CHANNEL.H
   - VXI_DB_MODULE.H
5. Network part not written yet

B.2 Code Modules

B.3 Program/Task Modules

B.4 Data Base Programs

B.5 Support Programs

B.6 Programs for Test of Software