REPRINTED FROM:

VMEbus IN RESEARCH

Proceedings of an International Conference
Zürich, Switzerland, 11-13 October 1988

edited by

C. ECK and C. PARKMAN
CERN

PARTICIPANTS EDITION

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

VMEAX– VAX with integrated VMEbus

Cord Eggers, Andreas Strabel et al.

IRTI
Ingenieurbüro für Rechnertechnologie
Hamburg, West Germany

Abstract: *The VMEAX-System is a collection of hard- and software modules implementing the idea to combine the benefits of two worlds:*

On one side the powerful VAX/VMS-System providing an arsenal of software development tools, and the VMEbus-System with its wide range of modules available from over 300 manufacturers on the other.

Each of the two systems is enjoying a lot of advantages over the other. So it is a good idea (at least we think it is) to combine these advantages. The resulting family of products then is VMEAX.

1. INTRODUCTION.

Those who are in some way or another concerned with computers of Digital Equipment Corporation (DEC) are appreciating the special characteristics of such systems. There is no need for us to discuss in detail all these features. They are sufficiently known we think. Some of the main points here to mention are (concentrating on VMS):

- an operating system of the first class
- a wide range of different machine sizes
- high reliability
- portability
- a big number of development tools
- presence of "high" software as
  - databases
  - CASE–tools
  - commercial kits

and so on ...

But as things are there are some flies on the ointment. The computers of DEC, formerly even more than today, are connected with technical applications. Even real–time applications were formerly established on DEC's PDP–11 systems, a possibility which rested upon two things: First the demands on real–time applications were lower than today, second there were operating systems like RT11 or RSX 11 which were not as comfortable as

VMS but had better response times. Today such applications on VAX/VMS systems are not longer feasible. Sure that depends on how you define the term "real–time". But if one considers response times in the nano– or micro–second range to be "real–time" one can easily prove VAX/VMS architectures to be unsuitable for such purposes.

On the other hand we have the VMEbus world which offers a wide variety of modules, a lot of which are meeting the requirements of real–time applications.

So we decided to couple both systems into a new one, now called VMEAX. (The missing "V" was optimized away!).

The main components of this coupling are the so called buscoupler on the hardware side, the IPK ("InterProzeß–Kommunikation") and some tools using the communication facilities of IPK on the software side.

2. SYSTEM OVERVIEW.

The following figure gives a rough view of a VMEAX–System:

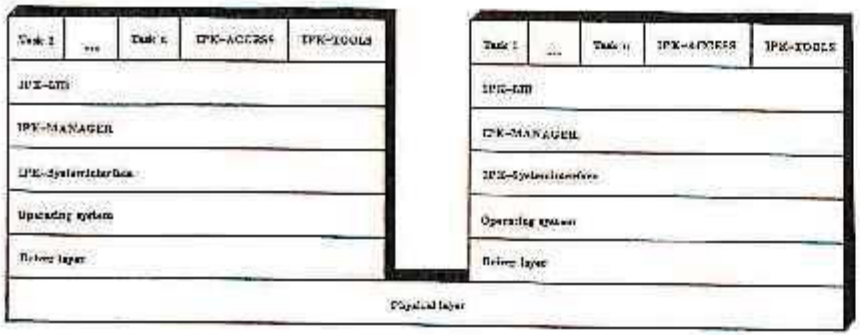

Fig. 1 System Overview

The figure shows the connection of two systems (typically VAX/VMS and VME/OS9) and the different layers which constitute this connection.

2.1 Hardware Layer.

The bottom layer represents the physical connection between the two systems. It is realized (or at least it can be realized) by the so called *Buscoupler*, a fast DMA device, which on the one side possesses an interface to the system bus (VMEbus, Q-Bus or Uni-Bus) and on the other side an interface to the Connector-Bus (C-Bus) which is inherent to the buscoupler. In order to connect two systems one needs therefore two buscoupler cards.

The main features of the buscoupler device:

- mutual transparency; registers of one buscoupler modul can be read from the other side and some can be written to. Semaphores prevent access conflicts. A protocol was developed which makes this exchange possible.
- data transfer takes place over a 64-poled twisted-pair cable with a maximal transfer rate of 4 Mbyte/sec (this reduces to 800 Kbytes/sec if you take system overhead into account).
- the buscoupler modules work asynchronously which makes each side independent from the other
- symmetrie of the buscoupler modules; therefore several connection schemes are possible:
  - VMEbus / VMEbus
  - VMEbus / UNIbus
  - Q-Bus / VMEbus
  - etc.
- microprogrammed control unit which simplifies the adaption to other bus systems.

See [1] for detailed information on the buscoupler.

2.2 Software layers.

2.2.1 Driver.

Next to the hardware is the driver layer. Drivers exist for RSX11, VMS and OS9 systems. They are designed to allow 64 channels to operate simultanously, have the structure of a multidevice interface and handle I/O tasks in an I/O queue, not necessarily in a sequential manner.

2.2.2 Communication, IPK and application design.

Equipped with buscouplers and drivers one already has the possibility to send and receive data from the other side. The only thing to do is to assign buscoupler channels (on both sides the same) and to write to or read from these channels respectively (With QIO calls in VMS for instance). This is the lowest level of communication possible. Although this gives you a really high speed connection it offers no comfort at all.

So something must be there to allow several processes to communicate with each other on controlled grounds. Besides this it should offer a number of useful services to each process. And it should be written in such a way that its use as a communication system is not limited to a buscoupled system of one VAX/VMS and one VME/OS9 system. Other communication devices should be possible and also more than two machines. To say it short it should have some networking capabilities and should offer some services which are typical for operating systems (eventing, semaphores and the like).

So we are at the very heart of the whole VMEAX System, the so called IPK which is an abbreviation for the German construction **InterProze3Kommunikation**.

Central to IPK is the idea of developing applications in a hardware independent manner, making it possible to concentrate on the actual logic of the application and only then to consider hardware questions. The principal steps in the realization of an application under the aid of IPK are:

1. *Application design*: in this stage the application is divided into a set of concurrent processes. No assumption is made as regards the actual machines on which these processes will be running. This decomposition in processes is (or at least should be) only determined by the division of the application in distinguishable parts as there are: collection , processing, compression, protection and exploitation of data. To stay independent of hardware you have to use logical names (for processes, channels, events and so on) instead of physical names.

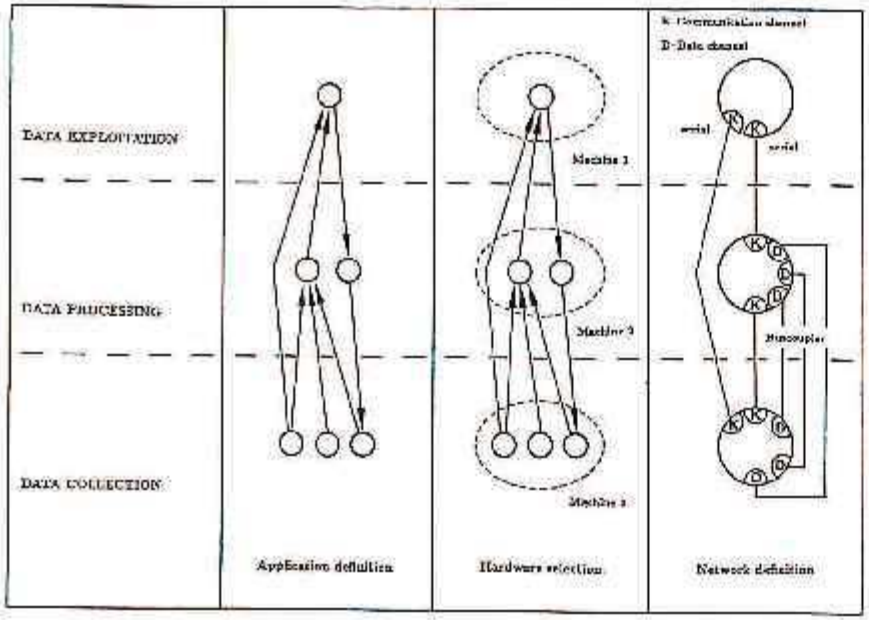

Fig. 2 Application design

2. *Net definition*: in this stage the collection of processes which constitute in some sense a "virtual machine" is to be mapped onto hardware, viz. you have to choose machines and communication devices and have to map the logical existing processes and communication pathes onto their physical counterparts.
3. *Test*: as always last stage is the test run. According to the type of errors and shortcomings you have to change the net definition or the whole application.

Basics of IPK

1) Logical Names
2) Process Net Concept
3) Communication Concept.
4) Software components.

1) Logical Names

To reach the goal of hardware independence the concept of logical names has been introduced into IPK. Only at run time a connection is made between logical names and physical entities. All logical names are of the *local* or *global* class. Local logical names are associated with a particular node. Global names are known to the whole application.

Here is a list of logical name types:

node
process
event
channel
semaphore

All types accept the following basic operations: (the parameter–lists are not complete; some commands require a *Password*, almost all permit the specification of a *timeout*–value, others require special parameters as *message–id* and so on.)

create ⟨logical–name⟩, ⟨ physical–name ⟩ ...
defines a logical name and its equivalence name;
delete ⟨ logical–name ⟩ ...
deletes a logical name;
link ⟨ logical name ⟩ ...
establishes a connection between process and logical name;
unlink ⟨ logical–name ⟩ ...
resolves this connection;
read ⟨ logical name ⟩ ...

returns status of logical name.

In addition to these basic operations there are other type specific operations. Here are examples:

Process – logical name:

connect ⟨ process–logical–name ⟩...
opens an IPK–session;
disconnect
closes an IPK–session;
run ⟨ process–logical–name ⟩...
Start process connected to logical name;
send_message ⟨ link–id ⟩, ⟨ message–address ⟩, ⟨ message–length ⟩...
send message to a process;
read_message ⟨ buffer–address ⟩, ⟨ message–length ⟩...
read message;
...

Event – logical name:

set ⟨ event–logical–name ⟩...
set an event to *true* (has occured) or *false* (has not occured);
get ⟨ event–logical–name ⟩... ⟨ destination–address ⟩...
read status of event;
wait ⟨ event–logical–name ⟩ ⟨ status ⟩...
wait until event has specified state;
...

Channel – logical name:

read_ch ⟨ channel–logical–name ⟩ ⟨ buffer–address ⟩...
read blocks of data of another process in a fast mode. In contrast to **read_message** there is almost no system overhead. This gives a high transfer rate but requires the processes to deal with all peculiarities (of data representation for instance) themselves.
write_ch ⟨ channel–logical–name ⟩ ⟨ buffer–address ⟩...
as **read_ch**, only other direction;
...

Semaphore – logical name:

lock ⟨ semaphore–logical–name ⟩...
lock resource exclusivly;
unlock ⟨ semaphore–logical–name ⟩...
unlock resource;
...

2) Process Net Concept

The networking capabilities of IPK are characterized by the following features:

a. Support of decentralized nets;

b. Communication channels;

c. Data channels (fast transfer);

d. Routing through half–dynamic routing tables.

3) Communication Concept

IPK supports two types of communication:

- exchange of short messages, mostly in ASCII and in greater intervals
- fast transfer of big amounts of data, mostly in binary form. IPK is nearly out of the game. Its only task is to provide a channel. The actual transfer is controlled exclusively by the interested processes. After transfer the channel is delivered back to IPK.

At the moment IPK supports the busconpler as a fast DMA–device and serial links for smaller data sets. Other communication devices are possible as is a connection to *Ethernet* or to the transport layer of an ISO/OSI–designed network.

4) Software components.

a. IPK–LIB

Contains the necessary routines to translate user commands into the IPK–internal format. Processes which want to use IPK must be linked with IPK–LIB.

b. IPK–MANAGER

The IPK–MANAGER constitutes the kernal of IPK. It is a process which must be started on each node participating in a net. The net configuration must be reported to each IPK–MANAGER via the IPK ACCESS utility.

c. IPK–ACCESS

Command line interpreter to communicate with the IPK–MANAGER (net configuration, logical names definition, collect statistical data etc.)

2.2.3 VMEAX–Tools.

Several tools have been developed using the IPK–Functions described above:

Virtual Terminal

The "Virtual Terminal" is a software tool which makes it possible to switch from VAX/VMS to VME/OS9 and vice versa. This switching is achieved

through pushing a defined key on the keyboard. This key is user definable.

Virtual disk

As a OS9 user you have the possibility to store data on a disk which in reality is a file on a VMS disk. You can even boot from this disk. As a consequence you can drive a complete OS9 system without an own disk and further you have with the VMS backup an automatic OS9 backup.

Development Tools

VMS users who wish to develop programs on a OS9 system but want to stay in the familiar VMS environment have several utilities (file copy, editing, starting C-compiler etc.) at their disposal. These utilities can be declared as "foreign commands" to VMS or they can be entered in the VMS command table. Syntactically these commands are similar to the respective VMS originals.

3. Concluding Remarks.

With VMEAX the user has the possibility to develop applications (insofar the VAX and VME worlds are concerned) to a large extent in a hardware independent manner. He (or she) has on one side the powerful VMS and the other side the "real-timeable" OS9 system. Those who do not want to use the whole of VMEAX can use parts of it as a development tool.

Further informations are available in:

Bibliography

[1] M. Seidel, "Wir haben die DEC-Welt mit dem VMEbus verbunden", *VMEbus* **2** (1987), 34–36.

(This article describes the internals of the buscoupler.)

[2] F. Hüller, "The best of both worlds", *DECKBLATT* **9**(1987)

[3] F. Hüller, "Die VAX mit dem integrierten VMEbus", *VMEbus congress, Munich 1987, congress volume*, 131–137

[4] A. Strabel, C. Eggers, "Interprozeßkommunikation auf verteilten Systemen", *VMEbus* **3** (1988) 34–36