EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
CERN — BEAMS DEPARTMENT

CERN-BE-2009-032 BI

Comparative VME performance tests for MEN A20 Intel-L865 and RIO-3 PPC-LynxOS platforms

M. Andersen, L. Jensen

Abstract

This benchmark note presents test results from reading values over VME using different methods and different sizes of data registers, running on two different platforms Intel-L865 and PPC-LynxOS. We find that the PowerPC is a factor 3 faster in accessing an array of *contiguous* VME memory locations. Block transfer and DMA read accesses are also tested and compared with conventional single access reads.

Geneva, Switzerland
November, 2009

1. Introduction

The existing VME CPU platform of which several 100's are installed, is becoming obsolete and the long-term replacement strategy means using Intel based machines running Linux. In order to check the performance of the new VME CPU modules we have compared the two platforms. These tests were done to ensure no performance degradation will result and that the involved hardware modules work.

2. Methods used

Different VME reading were done both with respect to different VME driver routines available, amount of read data, and the platform.

2.1. Software driver options

The VME modules' read access was been tested using 4 different methods. 3 of them covered single memory accesses while the last two used Block Transfer and DMA.

1. DriverGen II – IOCTL using a specific register function (ex. *BdiBobrGetidentifier()*)
2. DriverGen II –IOCTL using the Driver Access Library (DAL)
3. Direct user space mapping using *vme_map()* library function.
4. Block transfer using *vme_dma_read()* library function.
5. DMA transfer

2.2. Hardware modules used

The tests were performed using VME modules developed by BE-BI (BDI_BOBR and BI_BBQDAB). The latter one is one of the very few BI VME modules where the block transfer capability is implemented.

2.3. Register sizes and depths

The following table summarises the registers read during the VME tests.

Module	Register name	Register depth (bytes)	Register size (hex)
BDI_BOBR	Identifier	2	1
BDI_BOBR	bst1MainData	4	0x100
BI_BBQDAB	PostMortem	4	0x80000

2.4. Front-End Computers

Our tests were done the following BE-BI-SW development front-ends:

L865 (MEN A20)	cfv-865-bidev9
PPC4 (CES 8065DD)	dleitst1

3. Results

In order to help better understanding, the results are presented in groups with respect to a specific aspect (i.e. register size, platform, software driver etc.)

3.1. DriverGen II variations

We did not find any significant performance difference between the specific register access library of Driver Gen generated package and the DAL methods.

Driver Gen function option	Access time for single register on L865 (µs)
Register specific function	3.1
DAL generic function	3.2

Driver Gen function option	Access time for single register on PPC$ (µs)
Register specific function	3.6
DAL generic function	4.2

On both machines different drivers have shown similar access times, L865 is seen to slightly faster than ppc4.

3.2. Direct *vme_map()* vs. Driver gen library

The test to see VME read speeds was also done to compare Driver Gen available drivers and direct user space mapping of the VME module registers using ***vme_map()*** library function. These comparisons are available for L865 platform only.

Register/Function	Reg. Depth	Access time for one memory location (µs)
bst1MainData/dgII	0x100	2.33
bst1MainData/vme_map()	0x100	2.32

As can be seen the average access times are nearly identical.

3.3. Register size

The average time for VME reads are decreased when larger chunks of contiguous memory locations are accessed from a single. This is clearly evident when comparing access times between single word Identifier Register and *bst1MainData* register, which is 0x100 locations long.

Register/Platform	Depth	Access time per memory location (µs)
Identifier / L865	1	3.1
Bst1MainData /L865	0x100	2.3

Register	Depth	Access time per memory location (µs)
Identifier /PPC4	1	3.6-4.2
Bst1MainData /PPC4	0x100	1.0

The higher average access time can be explained by the overhead required to come from user-space to kernel (driver) space.

3.4. VME Block Transfer and DMA

In this section we present results of testing VME reads for a very large register using two independent performance enhancing features - VME Block Transfer and DMA (Direct Memory Access.) The two methods are similar in principle.

3.4.1. Block transfer

The VME BLock Transfer (BLT) means that VME data transfer between the module and the CPU is optimized for larger chunks of data, thus removing overhead. The optimization takes place on the VME controller/bridge level so that each that VME bridge does maintenance, notifications only for every 256 words transferred as opposed for every word being transferred. In addition less overhead is produced on the VME bus itself, where the protocol is optimized for 256 byte chunks. In essence the VME bus is blocked for the time of transfer of the 265 bytes, allowing higher priority transfers to take place.

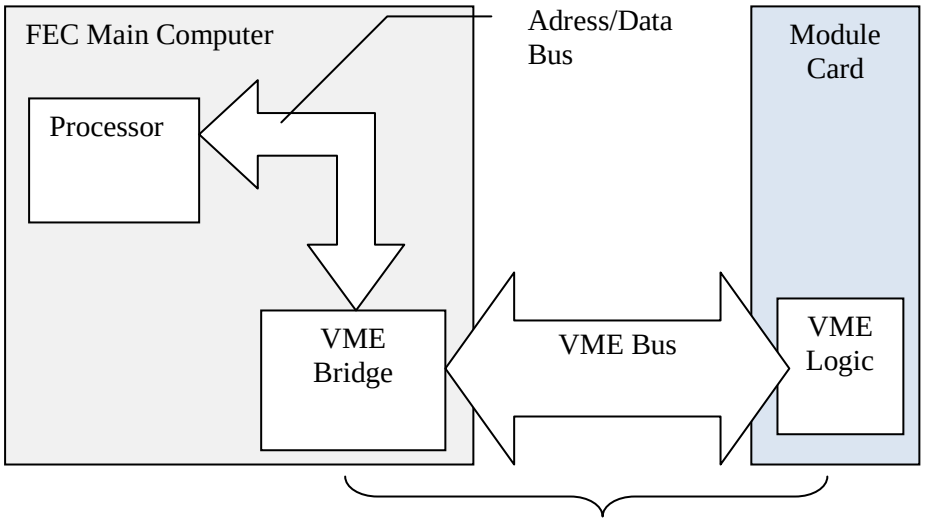


Figure 1 VME Block transfer

3.4.2. Benchmarking Block Transfer

It has to be noted that testing VME Block Transfer could only be done on L865 platform and using one of the very few supporting modules - BI_BBQDAB.

Register/ Method	Depth	Access time per memory location (µs)	Bandwidth(MB/s)
Normal VME access	0x80000	2.42	1.56
Block transfer access	0x80000	0.47	8.11

It can be seen that the Block Transfer increases the VME read speed by a little more than a factor of 5. It should be said that the maximum Block Transfer transfer-speed is determined here by the VME slave module through the use of dedicated VME control signals.

3.4.3. *DMA explained*

One of the main bottle-necks involved with reading values from the VME is related to the protocol between the main CPU and the VME Bridge. Instead of sending a large number of values for each VME access, the bridge implements a DMA mode which only requires setting-up. Direct Memory Access (DMA) is a method by which data from VME slave modules is copied directly from the "VME to PCI Bridge" bus controller into a buffer in user application memory, thus bypassing loading of a processor with instruction cycles for each memory location. In a non-DMA mode, the processor is executing copy instructions, which load data into local registers, when writing to each memory location. The DMA transfer bypasses the processor, taking data from peripheral IO (VME Bridge) directly to specified user memory locations. The main processor only has to issue a command to start copying, and receives notification while a whole chunk of data is already copied into the memory.

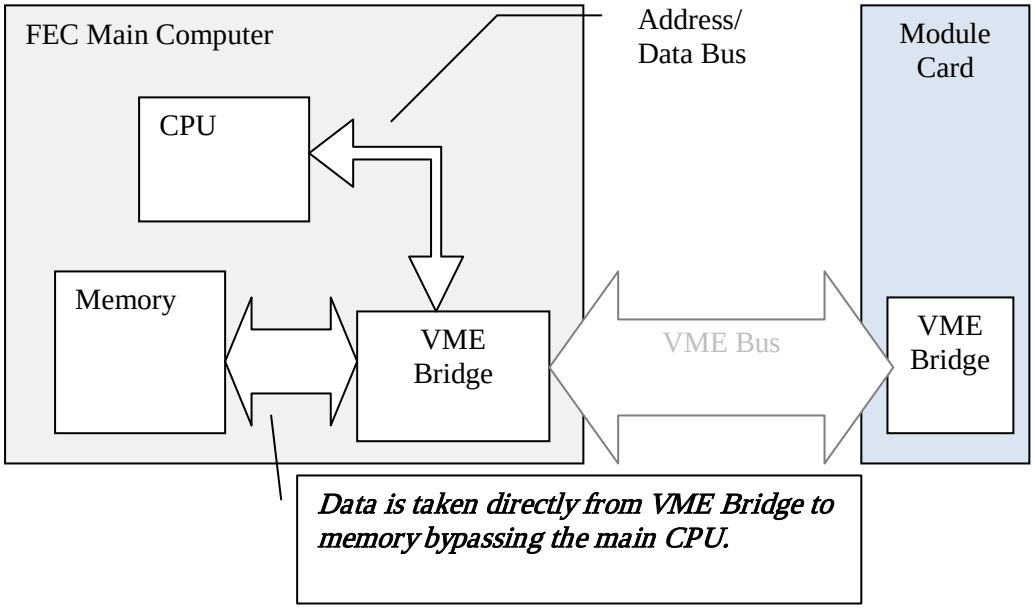


Figure 2 DMA Transfer

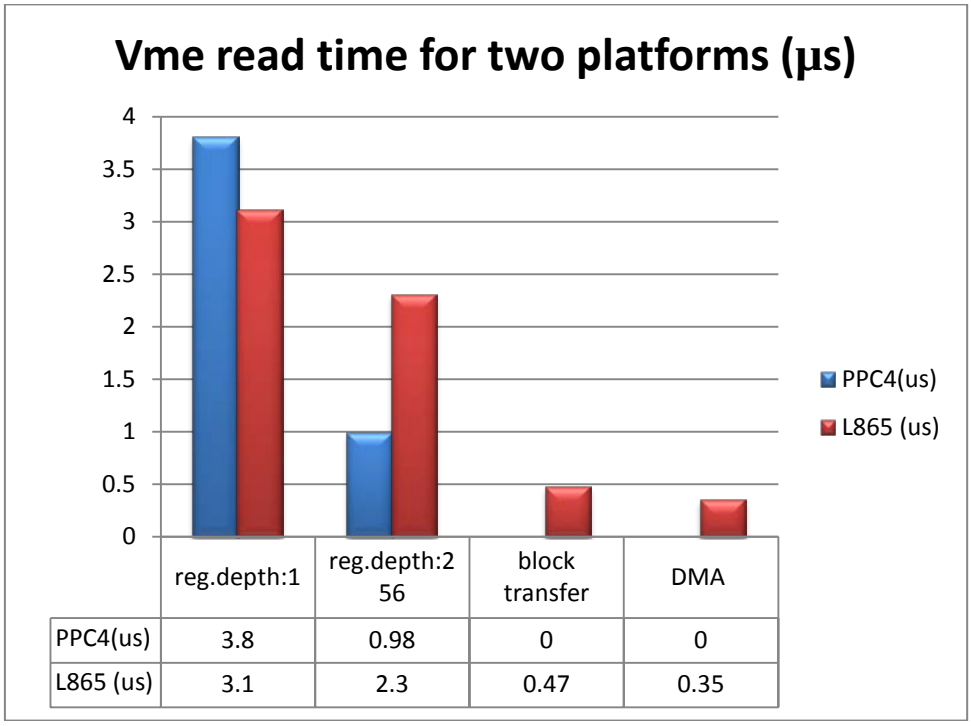
3.4.4. *Benchmarking DMA*

Register/ Method	Depth	Access time per memory location (µs)	Bandwidth(MB/s)
Normal VME access	0x10000	2.42	1.56
DMA access	0x10000	0.35	10.89

As can be seen DMA mode transfers allow gaining a factor 7 in speed. It is estimated that the time required to setup a DMA is around 2msec. It means that DMA transfers only make sense for reading above 1800 words. It is also worth noting that the DMA transfers have not been successfully tested on all modules and its usefulness must therefore be thoroughly tested in the lab before designing operational systems on it.

4. Conclusions

The bellow chart summarises the results of VME performance tests.



	reg.depth:1	reg.depth:256	block transfer	DMA
PPC4(us)	3.8	0.98	0	0
L865 (us)	3.1	2.3	0.47	0.35

From the performed benchmarks there can be derived several major trends:

- The speed of access of the software drivers used to fetch VME data is virtually the same. No real performance increase was seen by using direct mapping access *vme_map()* over Driver Gen II driver functions for reading larger blocks of memory. This would indicate that internally all software routines make the same number of execution steps when accessing VME memory.
- When accessing small registers, (i.e. registers having small length) L865 Linux platform is ~30% faster than ppc4 running LynxOS. The Linux platform averages at ~3µs while LynxOS has an average of ~4µs.
- When accessing registers of bigger lengths, both platforms perform better than on single registers. This is mainly attributed to smaller overhead due to fewer system/kernel function invocations. On Linux the performance improves by a factor of 1.33 (from 3.1 µs to 2.4 µs per access) while ppc4-LynxOS improves by a factor of 4 (from 4.0 µs to 1.0 µs per access), which leads us to the next conclusion.
- The ppc4-LynxOS platforms are much better at utilizing the spatial locality of the VME data. The factor 4 improvement between when accessing contiguous VME memory (i.e. buffers of bigger size) is explained by the kernel overhead coming from user-space to the kernel and back.
- VME Block Transfer is a good way of speeding up access to data of larger length. When comparing to access times without block transfer the improvement is more than factor 5(from 2.4 µs to 0.48 µs) However when comparing block transfer on Linux with contiguous memory access on LynxOS the improvement is merely a factor 2 (from 1.0 µs to 0.48 µs). It should be mentioned however that very few BI modules have block-transfer capability.
- DMA transfer has produced even bigger speedups than Block Transfer. A positive note on DMA is that it might be usable for most existing VME Cards since it uses the standard VME control signals (not the case for BLT).

5. To be investigated

The relatively slow 865-Linux VME conventional accesses to contiguous memory location is likely to present problems since many systems to be ported from LynxOS to Linux require reading of large blocks of memory in a relatively short time. VME slave modules should logically determine the time between consecutive accesses which does not presently seem to be the case for the Linux platform. Colleagues in BE-CO have been contacted to enquire whether this performance issue can be improved on in the short term.

Using Block Transfer and DMA optimisations can provide valuable performance increase for both existing and future projects. The amount of work to be done to introduce these features is to be determined closer. Combining Block Transfer and DMA in a single application is an interesting idea for very timing critical data transfers.