ATLAS Internal Note
ATL-D-ES-0004
Version 1.5
13 August 2003

Implementation and release notes for the VMEbus API on Linux based processors from Concurrent Technologies

Authors : M. Joos, J. Petersen

comments and queries to Markus Joos, CERN
+41 22 767 2364
Markus.Joos@cern.ch

Abstract

This note describes the implementation of the VMEbus API for the ATLAS ROD crate controller on single board computers from Concurrent Technologies under the Linux Operating system.

1 Introduction

This note describes the implementation of the VMEbus API for the ATLAS ROD crate controller [2] on single board computers from Concurrent Technologies [1] under the Linux Operating system.

2 System requirements

Hardware:

This implementation has been developed for Linux based SBC from Concurrent Technologies (CCT). The compatibility with the SBC families offered by CCT is as follows:

SBC type	Compatibility
VP-PSE	Not tested. Some features (e.g. BERR handling) will not work. Basic functionality should be available
VP-PMC	Fully tested by the authors
VP-CP1	Implicitly tested by e.g. the Pixel group
VP-100	Some test (mainly extended bus error handling) have been performed. No problems expected as the VMEbus interface of this card is very similar to VP-PMC
VP-110	Fully tested by the authors

Software:

The driver software has been developed on a 2.2.12 Linux kernel and ported to kernel version 2.4.9, 2.4.18 and 2.4.20. It has been tested with these kernel versions. The driver and library themselves do not require any kernel patches or extensions.

Required, however, is a means of allocating contiguous memory for internal use (DMA chain descriptors). This memory allocation has not been put into the VMEbus driver but outsourced to a separate driver & library: the cmem_rcc package. This package is described in a separate note [4]. The cmem_rcc driver has to be installed before VME_Open gets called for the first time. The services offered by cmem_rcc may also be used by the application programmer within the limitations of the implementation.

In addition the VMEbus library requires the io_rcc driver which is documented in [7]. This driver is used to map certain PCI registers into the virtual address space of the VMEbus library. As the cmem_rcc driver mentioned above io_rcc can also be used independently of the VMEbus library e.g. to implement a library for a PMC module.

3 Porting issues

Hardware:

Most of the source code of both the driver and the library should work as is on other VMEbus processors based on the Tundra Universe interface (e.g. Motorola, VMIC). Some modifica-

tions will however be required in the area of bus error handling and byte swapping which is based on some proprietary logic of CCT[1].

Software:

The authors have no plans for porting the package to other O/S platforms unless there is a justified request. Adaptations of the source code to more recent versions of the Linux kernel will be done routinely with the aim of keeping up with the latest kernel supported by CERN.

4 Endian issues

Being based on Intel CPUs the VMEbus SBCs of Concurrent Technologies are little endian machines. As VMEbus is big endian one has to solve the problem of byte swapping. The hardware of the CCT cards provides some logic for that purpose. Two bits in an I/O register allow to enable the byte swapping logic for the VMEbus master and the slave port. A third bit can be used to speed up the byte swapping under certain conditions. There are, however, some limitations and shortcomings:

- The byte swapping works for D8/16/32 single cycles and D32 block transfers. D64 multiplexed block transfers (MBLT), however, can not be swapped by hardware except for the VP-110.
- The fast byte swapping must only be used for aligned D16 and D32 transfers. Cycles from unaligned addresses do not work and there seems to be a general problem with D8 cycles which has to be further investigated.

The VMEbus driver and library itself don't have a notion of the byte swapping mode. The control bits of the byte swapping logic can either be initialized statically in the BIOS of the respective card (see [1] for details) or with the "vmeconfig" utility.

The recommended configuration is:

Master byte swapping: enabled

Slave byte swapping: enabled

Fast swapping: disabled

The actual setting can be monitored with option 4 (Show special registers) of "cctscope".

5 The standard utilities

The VMEbus API also requires three utilities to initialize the VMEbus interface, to help debugging the system and to scan the VMEbus:

5.1 The initialization utility

Access to VMEbus in general requires that one of the map decoders which are linking VME to the on-board bus (PCI) is programmed with the proper address offset and transfer mode (address modifier, write posting). Some VMEbus access libraries perform this function dynamically for each transfer. This is the most flexible method but costly in terms of function overheads. This Implementation of the VMEbus API for the RCC is based on a static initialisa-

tion of the map decoders which has to be programmed into the Universe chip (at boot time) by a utility program (see below). The limited number (8) of map decoders in the Universe requires that the VMEbus slaves be mapped into large windows instead of programming one map decoder for each VMEbus slave. The general rule should be to map the first module of each addressing type (-> AM code) at address 0x00000000. Further module of the same addressing type should be mapped to the lowest possible address (taking into account the granularity of the address decoder on the slave).

Vmeconfig must also be used to statically enable the interrupt levels that are to be handled by the SBC. As of Release 3 vmeconfig allows to define the type of interrupt (ROAK or RORA) per level.

The (static) initialization of the VMEbus interface is performed by the program “vmeconfig”. This program takes two command line parameters; the mode and the file name:

- The mode (“i” or “a”): The interactive mode (“i”) allows to prepare a configuration file for the VMEbus interface. If run in the automatic mode (“a”), vmeconfig reads the configuration file and initializes the VMEbus interface accordingly.
- The file name: This string defines the path and name of the configuration file to be used. The configuration file is not human readable and must be created or modified with the interactive mode of vmeconfig exclusively.

Vmeconfig contains some on-line help explaining its features and menus. It is nevertheless necessary to have some basic knowledge about the functioning of the Tundra Universe[5] interface and the address mapping on Intel based computers.

For the set-up of a master mapping the user has to enter the base address of a PCI address range which is to be mapped onto VMEbus. This address range must not be used by any other PCI device. An indication of the address ranges occupied by the other PCI devices can be obtained with the “lspci” utility program of Linux. Experience has shown that PCI addresses between the top of the memory and 0xe0000000 are usually available for the VMEbus interface.

An example configuration file (vmetab) can be found in the distribution together with the “vmeconfig” utility. Start the interactive mode (vmeconfig -i vmetab) and select option 2 (Dump all parameters) to display the set-up of the map decoders.

A recent addition to vmeconfig is the support for less frequently used address modifiers. It now can generate mappings for supervisor and program cycles in addition to the standard user / data mode.

5.2 The debugging utility

The program provided for this purpose is called “cctscope”. It is an interactive, menu driven program with built in help functions.

5.3 The scanning utility

The VMEbus can be scanned for slave modules with the “scanvme” program. Calling it in the form “scanvme -” displays the command line syntax and default settings.

6 File names and software organisation

The source code is contained in the package “vme_rcc“. The compilation of the library requires in addition header files from other DataFlow packages. The test programs have to be linked with libraries from a number of DataFlow packages. Please refer to the “use” statements in the “requirements” file of the “vme_rcc” package for details about interdependencies. All packages can be found in the DataFlow S/W repository (see http://atdaq-sw.cern.ch/cgi-bin/viewcvs.cgi/DAQ/DataFlow). As they are based on CMT it is necessary to install CMT locally. More information about the DataFlow code repository can be found at

http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/ROS/code_repository/

File name	**Remarks**
vme_rcc.o	This is the object file of the driver. It is to be installed into the Linux kernel
vme_rcc	A script for the installation of the driver. To be copied to /etc/rc.d/init.d and called from e.g. /etc/rc.d/rc3.d/S80vme_rcc
libvme_rcc.a	The library
/dev/vme_rcc	The device node
/proc/vme_rcc	The information file

7 Driver installation and trouble shooting

As mentioned in the introduction the use of the VMEbus library on a VMEbus processor from CCT requires the presence of three drivers: vme_rcc, cmem_rcc and io_rcc. The binaries of these drivers can be found in the respective DataFlow releases. The releases should contain versions for all supported kernel versions. In order to facilitate the system management all three drivers can be controlled by one script: drivers_rcc. This script (which is also distributed with the DataFlow releases) installs the drivers and creates the required device nodes. This installation of the script and the drivers consists of the following steps:

- copy drivers_rcc to /etc/rc.d/init.d

- create a link to drivers_rcc in /etc/rc.d/rcX.d with X being your run level (see /etc/inittab)

- create the directory /lib/modules/daq

- copy the drivers from the release to /lib/modules/daq

Once the drivers are installed you can get information about their state vie the special files “vme_rcc”, “cmem_rcc” and “io_rcc” in the /proc directory.

In case of a problem wit a driver you can restart it as “root”. Just type: /etc/rc.d/init.d/drivers_rcc restart_X with X being “vme”, “cmem” or “io”.

The debug output of the drivers (see below) can be enabled by restarting them with an additional parameter (debug=1). This is also handled by drivers_rcc. Type (e.g.) /etc/rc.d/init.d/drivers_rcc restart_vme_debug.

8 Debugging

The driver source code contains a large number of print statements for debugging purposes. They are conditioned by a preprocessor constant (-DVMERCC_DEBUG) defined in the driver's makefile. In order to activate the debug output you have to pass "debug=1" to the driver (see section above). The text messages are written to /var/log/massages. The flags VMEDEBUG and INITDEBUG are for expert use only.

The library has also been populated with debug statements which can be enabled with -DROS_DEBUG in the requirements file. The library writes its debug text to stdout.

9 Performance

The speed at which a VMEbus slave can be accessed depends on three parameters:

1) The speed of the slave's hardware. This parameter is mainly determined by the DS to DTACK delay of the slave VMEbus interface. It is more important for block transfers than single cycles

2) The speed of the master's hardware. Due to the complicated bridging from PCI to VMEbus the Tundra Universe does not reach the theorethical limits of the VMEbus protocol. The practical limits of the master interface are:

Cycle type	bandwidth in MB/s
Single cycle read (D32)	3
Single cycle write (D32) with write posting	13
D32 block transfer (read/write)	22 / 25
D64 block transfer (read/write)	45 / 55

3) The latency of the software. The overhead of calling a driver routine (context switch) is of the order of 1 us on a 700 MHz Pentium III. The overhead of the code itself differs from function to function. A typical value for a block transfer or a safe single cycle is 5..10 us.

10 Limitations of the hardware

The Tudra Universe chip does not support the read out of VMEbus FIFOs with block transfers. The problem here is that the Universe cannot keep the VMEbus address constant. The use of block transfers, however, is possible for FIFOs mapped into a larger address window (partial address decoding). For single address FIFOs it is possible to use constant address single cycles generated by the DMA controller. In both cases bus error terminated FIFO read operations only work if the FIFO contained an even number of 32 bit words. In case of an odd number the Universe loses the last word.

Due to problems with the host bridge it is not possible to use the VP/PMC or VP/PSE simultaneously as VMEbus master and slave.

11 Test programs

The vme_rcc package comes with a number of test programs. These were mainly intended as

debugging and verification aids during the development process of the driver & library. The source code of these programs, however, may contain some helpful examples for application programmers. The most generic program is “vme_rcc_test”. It allows the user to execute the functions of the API one by one in an interactive way. This feature could also be useful for some on-site debugging. More detailed instructions will be provided by the authors on a case by case basis.

The interrupt functions can be tested extensively using a test program (vmetrig_test1) which is part of the “CORBO” package[]. This test program relies on a CORBO VMEbus module[] as an interrupt source.

12 Block transfers

The library does not support all types of block transfers. Currently there is no support for some rarely used AM codes. The supported modes are defined as constants in vme_rcc.h. Currenly they are:

VME_DMA_D32W	Standard A32, D32, user, data, write
VME_DMA_D32R	Standard A32, D32, user, data, read
VME_DMA_D64W	Multiplexed A32, D64, user, data write
VME_DMA_D64R	Multiplexed A32, D64, user, data write
VME_DMA_A24D32W	Standard A24, D32, user, data, write
VME_DMA_A24D32R	Standard A24, D32, user, data, read
VME_FIFO_DMA_D32W	A32, D32, user, dara, write with constant VMEbus address (the data is transferred by means of single cycles under the control of the DMA logic)
VME_FIFO_DMA_D32R	A32, D32, user, dara, read with constant VMEbus address (the data is transferred by means of single cycles under the control of the DMA logic)

13 Application Program Interface

This section contains the implementation specific features and error codes of the individual functions. Functions not mentioned here are fully implemented as described in the API document.

VME_Open

Additional error codes

VME_ERROR_FAIL	Failed to open the “rcc_error” package
VME_FILE	Failed to open the device file (/dev/vme_rcc)
VME_CMEM_FAIL	Error from a call to a function of the CMEM_RCC library
various	VME_Open calls VME_MasterMap and may hence return any of the error codes of this function

VME_Close

Additional error codes

VME_FILE	Failed to close the device file (/dev/vme_rcc)
VME_CMEM_FAIL	Error from a call to a function of the CMEM_RCC library
various	VME_Open calls VME_MasterUnmap and may hence return any of the error codes of this function

VME_ReadCRCSR

Additional error codes

various	VME_Open calls VME_ReadSafeUChar and may hence return any of the error codes of this function

VME_WriteCRCSR

Additional error codes

various	VME_Open calls VME_WriteSafeUChar and may hence return any of the error codes of this function

VME_BusErrorRegisterSignal

Additional error codes

VME_BERRTABFULL	Bus error process table full (max. 10)
VME_BERRNOTFOUND	process has no bus error registered

VME_MasterMap

Mapping types

The VME_MasterMap_t structure passed to the VME_MasterMap function contains a field called “options”. It is used to specify the AM code for single cycles. The various modes are defined by constants in vme_rcc.h. You can combine the modes (e.g. “VME_AM_DATA | VME_AM_USER”)

The modes are:

VME_AM_USER	Standard user access
VME_AM_SUPERVISOR	Access with supervisor provilege
VME_AM_DATA	Standard access to data
VME_AM_PROGRAM	Special access to program code

In order to use these modes the map decodes of the Universe chip have to be initialized accordingly by “vmaconfig”.

Additional error codes

VME_NOMAP	Failed to find a free entry in the internal master map table. This means that too many master mappings have been opened
VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_NOSTATMAP	Error from driver. Requested mapping does not fit into static mapping of Universe (see vmeconfig)
VME_VIRT	Failed to allocate a user virtual address range for the mapped region
various	VME_Open calls VME_ReadSafeUChar and may hence return any of the error codes of this function

VME_ReadSafeUInt, VME_ReadSafeUShort, VME_ReadSafeUByte

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_WriteSafeUInt, VME_WriteSafeUShort, VME_WriteSafeUByte

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_MasterUnmap

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_MUNMAP	Error from munmap() system call

VME_MasterMapDump

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_KMALLOC	Error from driver. Failed to allocate memory

VME_BusErrorRegisterSignal

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_BERRTBLFULL	the table containing bus error links is full (# entries > VME_MAX_BERR_PROCS)
VME_BERRNOTFOUND	error on unregister (signal = 0): the pid of the user process is not found in the bus error table

VME_BusErrorInfoGet

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_SlaveMap

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_NOSTATMAP	Error from driver. Requested mapping does not fit into static mapping of Universe (see vmeconfig)
VME_NOMAP	Failed to find a free entry in the internal slave map table. This means that too many slave mappings have been opened

VME_SlaveMapDump

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_KMALLOC	Error from driver. Failed to allocate memory

VME_BlockTransferInit

Additional error codes

VME_NOSIZE	One of the size parameters is zero
VME_ALIGN	Misaligned address / size
VME_NOCHAINMEM	All chain descriptors are in use

VME_BlockTransferStart

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_NODOMEM	Error from driver. No space left in the internal table of completed transaction. This could indicate that a process does not call VME_BlockTransferWait
VME_ERESTARTSYS	Error from driver indicating that it got interrupted when it was waiting to get the DMA semaphore

VME_BlockTransferWait

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_ILL_TO	The value of the parameter “time_out” is invalid
VME_DMAERR	There was a VMEbus BERR during the execution of this transfer

VME_BlockTransfer

Additional error codes

various	This function may return any of the errors of VME_BlockTransferInit, VME_BlockTransferStart, VME_BlockTransferWait and VME_BlockTransferEnd

VME_BlockTransferDump

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_KMALLOC	Error from driver. Failed to allocate memory

VME_InterruptLink

Additional error codes

VME_TOOMANYHDL	too many handles (max. = 10)
VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_ENOMEM	Error from driver indicating problem with kmalloc

VME_InterruptReenable

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user
VME_LVLISNOTRORA	the level is not of type RORA

VME_InterruptWait

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_InterruptRegisterSignal

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_InterruptInfoGet

Additional error codes

VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_InterruptUnlink

Additional error codes

VME_ILLINTHANDLE	illegal handle: the number of associated vectors = 0
VME_EFAULT	Error from driver indicating problem with copy_from_user() or copy_to_user

VME_InterruptGenerate

Currently NOT implemented

VME_InterruptDump

Currently NOT implemented

The current release of the driver and library contains an number of additional service functions which are needed by the tool programs (e.g. vmeconfig, cctscope). They are not meant to be called by the user and may be outsourced to a dedicated driver in the future.

14 Application programming

The VMEbus library uses an external package for the formatting and reporting of errors: “rcc_error”. It is therefore necessary to include “rcc_error.h” into programs using the VMEbus library and to link the rcc_error library into the application. There is no specific documentation for rcc_error. Programmers are asked to refer to [6] which describes the iom_error package. The packages iom_error and rcc_error are basically only different by the prefix (iom / rcc).

15 Release history

15.1 Release 1 (1. Feb. 2002)

This is the first official release. It implements most of the functions specified in the API document and has been tested under Linux 2.2.12.

15.2 Release 2 (8. Mar. 2002)

This release adds support for Linux 2.4 kernels (tested with 2.4.9) and has some bug fixes

15.3 Release 3 (14. Jun. 2002)

This release provides support for RORA type interrupts as requested by several users. As a consequence the API of some of the interrupt functions has changed.

Via the vmeconfig utility, the user can define the interrupt levels to be either disabled or enabled and, in the latter case, if the level is used for **ROAK or RORA** interrupters. In the case of RORA interrupts, the driver disables the interrupt level and the user has to re-enable the interrupt in the application after the interrupt has been serviced.

In this release the VMEbus driver contains some undocumented ioctls() and functions for PCI address mapping and I/O access. These functions are required by some of the applications it the package (e.g. cctscope) but are not meant to be used elsewhere. They will be removed and put into a separate driver for Release 4.

15.4 Integration into the DataFlow repository (7. Oct. 2002)

Since October 2002 the package vme_rcc (and all the other packages mentioned in this document) are stored in the DataFlow code repository. There are no “stand-alone” releases of the RCC software any more. Since release 3 there have been several additions (e.g. user defined AM codes and support for SYSFAIL) as well as some bug fixes. Details can be found in the release notes of the respective DataFlow releases as well as in the “cvs log” of the source files.

16 References

[1] Concurrent Technologies, Technical Reference Manual for the VP PMC/P3x VMEbus Pentium III Singe board Computer, Manual code 550 0010-50 Rev. 01

[2] http://edmsoraweb.cern.ch:8001/cedar/doc.info?document_id=325729&version=1

[3] http://atddoc.cern.ch/Atlas/Notes/136/Note136-1.html

[4] https://edms.cern.ch/document/336290/2

[5] http://www.tundra.com/page.cfm?TREE_ID=101023

[6] http://atddoc.cern.ch/Atlas/Notes/051/Note051-1.html

[7] https://edms.cern.ch/document/349680/1