GENERIC VME INTERFACE FOR LINUX 2.6 KERNELS

A. Homs#, F. Sever, ESRF, Grenoble, France

Abstract

From the beginning of the ESRF both the machine and the beamline control instrumentations were based on VME bus diskless crates equipped with Motorola CPU boards running OS-9. Several modernization steps were performed to migrate from OS-9 to Linux running either on the VME CPU or on a remote industrial PC, connected to the crate using PCI/VME bus coupler. An initial implementation of a generic VME driver interface was developed for Linux 2.4 which allowed the same VME driver code to work on the different platforms. This paper presents the complete re-writing of the above VME layer to fully conform to the abstract bus/device interface provided in Linux 2.6 kernel. The new subsystem separates the rolls of the VME hosts, controlling the target bus, and VME devices, using generic bus functionality exported by the hosts. It also features safe hot-plug device detection and removal in SMP systems. The drivers for the bus controllers and VME boards used at the ESRF were ported to this new structure.

ESRF CONTROL SYSTEM EVOLUTION

Original System

The first implementation of the ESRF machine and beamline (BL) control system used diskless VME crates controlled by Motorola CPU boards running OS-9. The low level hardware access was provided by device drivers, composed by OS-9 modules and their corresponding libraries. The board functionalities were exported to the network through TACO, a client/server control architecture based on RPC developed at the ESRF. The device servers execute the commands triggered by the TACO clients running on HP or SUN workstations. Typical clients in the machine control system are specific Graphical User Interfaces (GUIs), while for the BLs the main experiment application is SPEC, a command line interface (CLI).

Evolution on the Beamlines

The VME CPU card was replaced by the SBS Bit3 PCI/VME bus coupler, directly connected through a fibre optic link to an industrial PC [1]. It allows to control the boards on one or several remote VME crates as if they were connected locally, transmitting IRQs and allowing DMA transfers. The operating system was SuSE/Linux 7.2 running on dual Pentium III CPUs at 1 GHz.

Evolution on the Machine

On the machine side it was decided to keep local CPU card in the crate. In order to increase hardware performance, the Motorola CPU board was replaced by a more powerful Intel CPU board (VP101 from Concurrent Technologies) which includes Tundra Universe II PCI/VME bus bridge. The chosen OS was Debian/Linux 3.0, providing better support to diskless systems.

In addition, to the VME instrumentation, PCI and cPCI hardware was also added on both the BLs and the machine, either by using PCI/cPCI bus extenders on industrial PCs or CPU boards in cPCI crates. However, the PCI instrumentation is out of the scope of this paper.

Software Implementation

Linux 2.2 and 2.4 kernels were initially used. The VME driver interface was written to allow the same card drivers to work in both hardware configurations [2]. However, their binary code was highly dependent on the bus-coupler implementation due to heavy use of C-macros, making the code notably cryptic. The software layout for Linux 2.4 drivers is shown in Fig. 1.

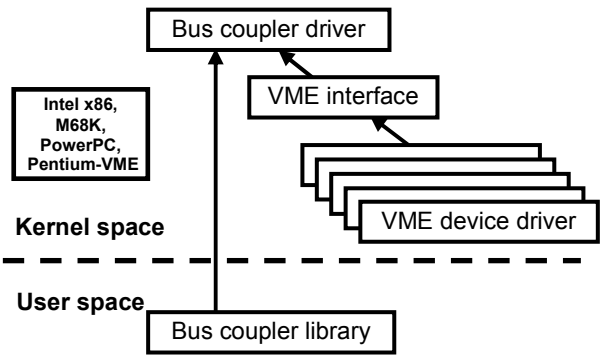

Figure 1: VME software layout for Linux 2.4

VME STRUCTURE IN LINUX 2.6

With the introduction at the ESRF of Linux 2.6 and its new device model [3] the VME interface layer, now called VME core, was completely redesigned. A new bus type, *vme*, was created. Each *vme* bus in the system is controlled by a *vme_host* and is used by one or more *vme_devices*. The layout of the new VME subsystem drivers is shown in Fig. 2.

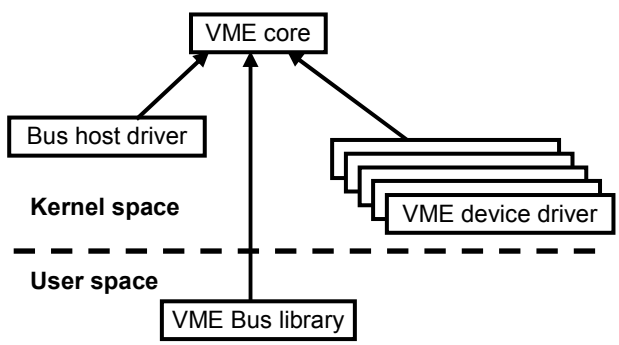

Figure 2: VME software layout for Linux 2.6

In contrast to the Linux 2.4 implementation, the VME card drivers are completely independent of the bus (host)

#ahoms@esrf.fr

controller. The connection between *vme_device* and *vme_host* is done via generic *vme_host_operations* including probe and regular read/write cycles, *vme_region* memory mapping and IRQ management (register, mask). In the same way, VME devices are automatically created and managed by means of a *vme_driver* structure, specifying predefined I/O base addresses and IRQ vectors as well as the corresponding *file_operations*. This model is used in the other Linux kernel subsystems (PCI, USB, SCSI, etc.), and it provides the maximum flexibility in terms of compilation, run-time execution and long term code evolution.

The two types of VME controllers used at the ESRF, the SBS Bit-3 and the Tundra Universe II PCI/VME bus bridges, were successfully ported to this architecture. From the VME device side, the ESRF VPAP (motor controller) and VCT6 (counter/timer), the Compcontrol CC133 (encoder), and the ADAS ICV196 (digital I/O), ICV150 (ADC) and ICV712/6 (DAC) were also ported to the new structure.

Features of the New VME Subsystem

As mentioned before, a *vme_host* gives access a VME bus. In the case of multiple SBS Bit-3 PCI/VME bus couplers in the same PC, independent VME buses (*/dev/vme0*, */dev/vme1*, …) will be created in the system. For each existing bus, the registered *vme_driver* will probe for present boards at the predefined I/O addresses. This operation is automatically done when a new *vme_driver* is registered, or when a new *vme_host* is available, emulating a *hot-plug* behaviour not supported at the hardware level.

In order to be completely "*hot-plug* compliant", when the PCI/VME bus coupler is disconnected or the remote VME crate is switched off the bus is disabled and the corresponding *vme_devices* are automatically deleted. A kernel thread is responsible of polling the status of the available *vme_hosts* and triggering the probe for devices if the bus becomes active again.

The system fully integrates the /sys filesystem (*sysfs*) by exporting the bus/host resources (memory maps, IRQs) and the device info (configuration and IRQ statistics). In addition *vme_class_devices* are also created, exporting the device major-minor numbers used by the *udev* daemon for creating the user-space entry nodes (*/dev/vpap_01*, …).

As shown in Fig. 1, generic access to a VME bus from user space in Linux 2.4 was only possible through the bus controller driver library, which in general has a manufacturer specific interface. Besides, the VME drivers were not aware of concurrent access through this library. The Linux 2.6 implementation provides a generic VME access library (Fig. 2), allowing the development of "user-space" drivers that are handled in the same way than VME kernel drivers. Locking mechanisms are provided to the *vme_host* driver to avoid concurrent access to the bus by users of its specific library.

A major feature of the VME subsystem is the support of multi-CPU systems (SMP) and multi-threaded applications. Special care was taken during its design to ensure that all the operations are "*hot-plug* safe", using the appropriate kernel locking mechanisms (*spin_locks*, *semaphores* and *kobjects*). In particular, a *vme_device* can not disappear if a program is performing a non-blocking *read/write/ioctl* operation in its driver. If it is a blocking operation (*vme_wait_event*), the device can be deleted and the system call fails with the corresponding error notification. In addition, if another thread was also blocked waiting for the first thread to release the device (*vme_down*), it will also be notified that the device no longer exists.

Finally, a binary interface version checking ensures that the *vme_host* driver, the *vme_driver* and the user-space driver will communicate to the VME core in a consistent way. This allows the controlled evolution the three interfaces when distributing binary packages.

Installation Status

The new VME architecture has been introduced on the BL and machine control systems since the first half of 2007. The current installation status is shown in Table 1.

Table 1: Current installation at the ESRF

	Linux 2.4		**Linux 2.6**	
	VME crates	**Industrial PCs**	**VME crates**	**Industrial PCs**
Beamlines	63	49	15	11
Machine	49	5	5	59

FUTURE STEPS

- Move the project to Sourceforge.net
- Implement generic DMA access
- Integration into ESRF fast acquisition architecture [2]
- Port to recent kernels (currently runs on 2.6.9)

ACKNOWLEDGEMENTS

Have also participated in (previous stages of) the development of this project: R. Hirst, S. Marguet, D. Kimdon, M. Pérez, M.C. Domínguez, E. Papillon, D. Beltrán, A. Beteva, P. Fajardo, J. Klora, A. Götz, P. Mäkijärvi and B. Regad.

REFERENCES

[1] A. Götz, A. Homs, B. Regad, M. Pérez, P. Mäkijärvi, W-D. Klotz, "Modernising the ESRF Control System with Gnu/Linux", Proceedings of ICALEPCS-2001, San Jose, California, pp. 325-327 (2001).

[2] A. Homs-Purón, D. Beltrán, A. Beteva, M.C. Domínguez, P. Fajardo, A. Götz, J. Klora, E. Papillon, M. Pérez, "Linux/PCI: The ESRF Beamline Control System Modernization", Proceedings of ICALEPCS-2003, Gyeongju, Korea, pp. 172-173 (2003).

[3] J. Corbet, A. Rubini, G. Kroah-Hartman, "Linux Device Drivers", 3rd edition, O'Reilly (2005)