SIMPLE CAN-BUS ADAPTER FOR ACCELERATOR CONTROL RUNNING UNDER LINUX AND RTLINUX.

F. Nedeoglo, D. Komissarov, O. Novozhilov, Department of Physics, Moscow State University,
A. Chepurnov, Institute of Nuclear Physics, Moscow State University, 119899, Moscow, Russia

Abstract

CAN-bus was chosen as basic fieldbus for newly designed distributed control system for electron linac. To provide CAN-bus access for PC CAN-bus ISA compatible adapters have been designed. The adapter is based on Philips SJA1000 CAN-bus controller and provides fast access to CAN-bus through direct memory mapping. To use the adapter under Linux the Linux kernel mode driver was developed. To support operation in real-time driver for RTLinux has been developed and tested. Both drivers provide the same POSIX IO compatible interface for application software.

1 INTRODUCTION

When we decided to use CAN-bus in new control system [1], we faced the problem what type of CAN-bus adapter to use right away.

We use Linux together with RTLinux extension as developing and runtime environment for our control application, so we were need in CAN-bus adapter which is supported under Linux [2,3]. That time, two years ago, we didn't find on the market CAN-bus adapter which would be supported under Linux. So, we decided that if it is necessary to develop our own Linux driver, it is reasonable to develop our own hardware CAN-bus adapter too to make it construction completely transparent for our programmers.

As a result ISA-compatible passive ISA to CAN-bus adapter was designed and manufactured. The set of chips was so compact that next step was made when two independent CAN-bus adapters were placed on the single half-size ISA board. It allowed access two independent CAN-bus channels through single ISA slot of PC's motherboard.

2 HARDWARE STRUCTURE OF CAN-BUS ADAPTER.

Philips Semiconductor SJA1000 CAN-bus single-chip controller was selected to implement hardware CAN-bus protocol [4]. To control SJA1000 through ISA bus from PC it is necessary to demultiplex multiplexed data-address bus of SJA1000. Memory–map technique was used when control registers and message FIFO of SJA1000 are reflected in selected block of memory through ISA-bus. Interrupt driven and polling mode of operation are possible. Polling mode wastes PC resources very seriously, so interrupt driven mode of operation is preferable. CPLD with very useful "In Circuit Programming via JTAG interface" feature is used to organise interface between multiplexed parallel bus of SJA1000 and ISA-bus. We use (and recommend to do it especially for particle accelerator and industrial applications) galvanic isolation between PC and CAN-bus physical line. "Transmit" and "receive" lines of SJA1000 are isolated via fast opto-coupler of the 6N137 type. Isolated area with CAN-transiver (PCA82C200 type) is feeded with +5V power voltage through Burr-Brown DC-DC converter DCP0505 with 1500V electrical isolation.

3 SOFTWARE FOR CAN-BUS.

3.1 Support of CAN-bus adapter under Linux.

To make CAN-bus alive under Linux, Linux kernel driver for CAN-bus adapter was developed together with simple Monitor program. The Monitor program listens CAN-bus, visualises traffic of CAN-bus and allows user to send manually composed message over CAN-bus network. The driver can handle up to four CAN-bus-ISA adapters simultaneously.

Application software interfaces with the driver by means of writing and reading CAN frames to or from the special character device file.

The driver supports asynchronous mode also. In that mode driver sends the signal (SIGUSR2) to the user process, when a CAN-frame arrive from CAN-bus network. Therefore it is possible to build the interrupt driven application to work with CAN-bus under Linux.

3.2 DeviceNet protocol stack under Linux.

To ensure high level of compatibility and ability to use as home made as "off the shelf" components DeviceNet high level protocol for CAN-bus was selected as basic for our CAN-bus application programs [5]. We did not find DeviceNet software stack under Linux. So DeviceNet compliant protocol stack was developed [6].

The DeviceNet compliant protocol stack provides Slave capabilities for different types of front-end controllers and Master capabilities for host personal computer (PC) running under Linux OS.

The protocol stack was developed in the form of software library (Figure 1) in order to port it to different hardware platforms. It consists of the following components:

- the library kernel,
- the module with system dependent functions,

- the module with interface to CAN-bus.

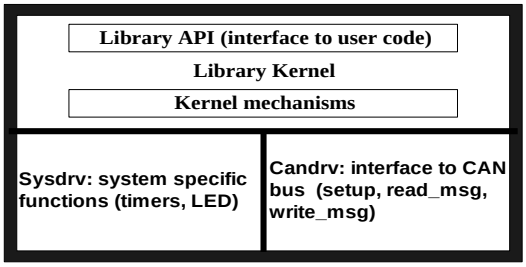

Figure 1. The structure of the DeviceNet library.

This partitioning scheme ensures portability of our software to platforms with poor resources (micro-controllers) as well as platforms with rich resources such as PCs. The library kernel contains the only protocol stack and interface to the application programs. All dependencies of particular environments are located in the two other parts of the library. So if some kernel module has been debugged and tested under Linux, it can be used in DeviceNet compliant devices, developed for other platforms. To port the library to a new platform it is necessary to modify system dependent functions (module called *Sysdrv*) and interface to CAN-bus (called *Candrv*).

Various versions of DeviceNet library have been tested including Intel-compatible PC under Linux, single-chip micro-controllers from Microchip (PIC16C7x/87x) and DSP from TI (TMS320C2xx).

3.3 Software support of real-time.

Real-time extension of Linux OS - RTLinux allows developing software components, which have got hard real-time capability [7]. The real-time processes are implemented in RTLinux as lightweight threads and run in the kernel memory space. RTLinux coexists with Linux OS, Linux kernel operates as separate real-time process with the lowest priority using a virtual machine layer in RTLinux.

We use the latest stable version 2 of RTLinux. The version 2 consists of core component and several optional components. The core component is distributed as Linux kernel patch. The core component allows registering the low interrupt handlers that cannot be pre-empted by Linux itself. The optional components provide:

- pure priority based scheduler,
- set of functions to work with system clock and timers,
- support for POSIX I/O interface (read/write/open/close) for real-time device drivers,
- real-time FIFOs, that connect a real-time process and Linux user space process through a special character device file so the Linux process can read/write to real-time component,
- shared memory between real-time components and Linux processes.

RTLinux uses the loadable kernel module mechanism implemented in Linux OS to load as real-time processes as optional components into the memory.

RTLinux version 2 supports the real-time POSIX.1b threads as well as API of RTLinux version 1 for backward compatibility. There is also support for POSIX mutex locks in the latest minor versions of RTLinux 2.x. The support of POSIX IO interface in RTLinux provides a filesystem like interface to real-time drivers.

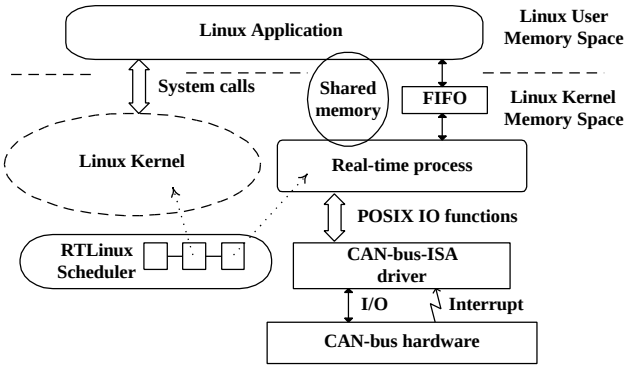

Figure 2. Interaction of RTLinux components in control system software.

RTLinux provides only basic real-time capabilities whereas Linux OS provides all other general services. An application that requires real-time capabilities consists of two parts: real-time kernel module implementing real-time functionality and Linux process communicating via FIFO or shared memory with the real-time module.

To support access of real-time software running on front-end PC to CAN-bus the RTLinux driver for CAN-bus ISA adapter was developed. The driver provides POSIX I/O interface for real-time control and acquisition processes.

If real-time process tries to access CAN-bus it opens special character device file (/dev/canX), and just reads from or writes to this file the CAN frames (Figure 2).

3.4. Support of CAN-bus adapter in other operating systems.

Unfortunately it is rather difficult to achieve unified environment for development and run-time. So, we should use Windows based software tools when we develop embedded hardware devices. We have got software development tools for DSP and micro-controllers (C and assembler compilers, TAG debuggers and so on) under Windows only and there is no such kind of tool under Linux yet.

So it was necessary to supply developers of embedded hardware with the possibility to debug CAN-bus compatible components within one PC.

So, system Windows NT driver was developed together with Windows Monitor application. Driver support up to four independent CAN-bus adapters installed in one PC.

The Monitor program allows to watch CAN-bus activity and to send message over CAN-bus. CAN-bus line parameters could be changed manually.

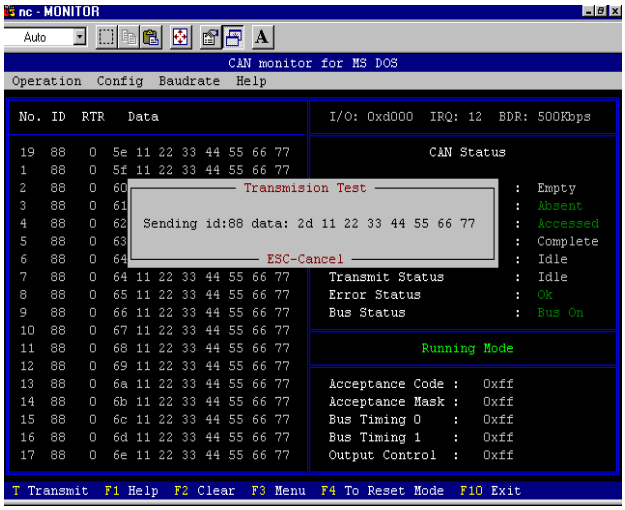

Figure 3. Screenshot of CAN-bus Monitor program running under Windows95

Monitor program with the same functions was developed for MS-DOS too. It allows to use old fashion PC as test stations running Monitor program as DOS application under Windows95.

4 CAN-BUS BASED CONTROL SYSTEM

4.1 General layout of control system.

The control system consists of two classical levels (Figure. 4) - non-real-time top level and real-time front-end level. Both levels use Intel-compatible PCs.

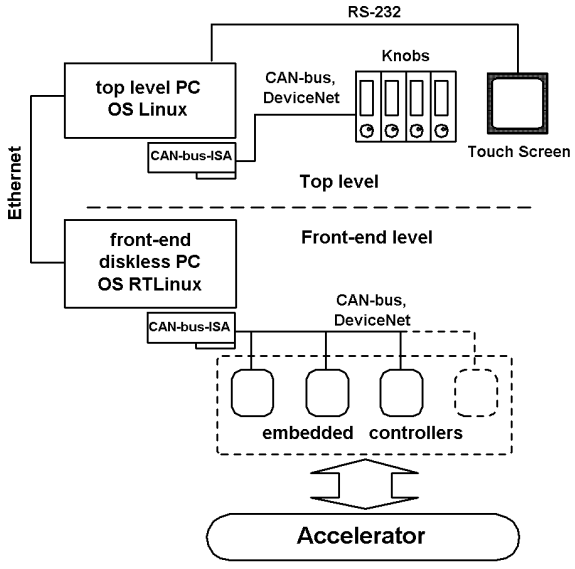

Figure 4. Layout of the control system.

Front-end level supports fast control algorithms, hardware locking, fast feedback loops, and signal conditioning hardware. The basic functions of top level are bootloading of front-end computers, supporting of man-machine interface and providing necessary database capabilities.

Top level and front-end PCs communicate via Ethernet fibre optic link to provide galvanic isolation between levels of control system.

4.2 Top level.

PC compatible computer runs under Linux 2.2.x at the top level of control system. To improve interaction between operator and accelerator we use knobs-type modules and plan to use touch screen. The module constructed around single-chip micro-controller consists of encoder, two lines of high brightness LCD, and four keys with corresponding LEDs. The modules (up to four in our case) could be assigned dynamically with any adjustable or controllable parameter of accelerator. The modules communicate with top level PC via CAN-bus.

4.3 Front-end level.

At the front-end level the diskless PC runs under Linux 2.2.x with RTLinux 2.2a (real-time extension of Linux). Diskless PC boots operating system via BOOTP protocol from the top-level PC. Then root file system is mounted with the help of NFS protocol.

CAN-bus-ISA adapter installed in front-end PC controls embedded controllers that belonged to a family of “Smart Devices”--intelligent controllers which support functions of real-time digital feedback control, data acquisition and processing [8].

4.4 Simple CAN-bus protocol.

DeviceNet protocol stack was not completely finished when control system should been start working. So the simple CAN-bus protocol was developed under Linux to use temporary to start up new control system and test components.

During the test and tuning of control system we developed simple, master based CAN-bus protocol. The interpretation of CAN ID field [9] by our protocol is shown at Figure 5. The upper six bits of CAN ID field contain slave’s ID. Following 2 bits are interpreted as message type, and lower 3 bits are reserved for future use and should contain zeroes.

One master device (PC) controls many slave devices (embedded controllers). To distinct slaves one from another, the slave is assigned by unique identificator. Master device processes all of the frames arriving from CAN-bus, and slave processes the only frames, which CAN ID contains ID of appropriate slave. Any frames transmitted by slave into CAN-bus network should contain slaves ID in CAN ID field. As master device operates in broadcast way, it is not necessary to assign an identificator to master device. When master device

intends to transmit frame to appropriate slave device, it includes the slave's ID into CAN ID field of the frame.

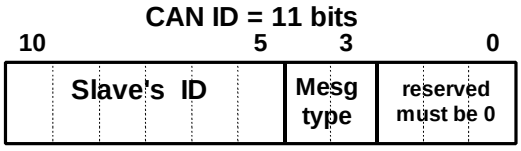

Figure 5. Interpretation of CAN identificator

Four types of message: start, stop, auto and generic are utilised in data exchange. The start-, stop- and auto-messages are dedicated to slave's operation in automatic mode. Master sends start frame to appropriate slave device at this mode of operation. Then the slave starts to transmit messages of auto type in endless loop. To finish this transmission master sends the stop message to the appropriate slave. Generic type of messages is used to organise the exchange messages between master and slave on demand of master device.

4.5 Control system application software

The application software of control system is based on architecture with Distributed Shared Memory (DSM) [3]. Modules of application software watch and control an accelerator through a segment of DSM. Mirroring mechanism of DSM segments is hidden from application software. Software components accessing the segment of DSM, and not responsible for mirroring, might know nothing about inter-level communication construction and were not concerned with the appearance of data. This approach ensured rather clear application program interface, which simplified work of programmers and made possible the independent development of parts of application software as mirroring algorithm for different types of hardware.

5 CONCLUSION

CAN-bus ISA-compatible adapter together with accompanying software were used successfully as for development of CAN-bus compatible devices, as for development of control software applications as for run of control system of particle accelerator.

6 ACKNOWLEDGEMENTS

The authors acknowledge the Organising Committee of PCaPAC for financial support to participate the conference.

REFERENCES

[1] A. Chepurnov, A. Alimov, et. al., "Control System for New Compact Electron Linac.", // Proc. of ICALEPCS'99, Trieste, Italy, ISBN: 88-87992-00-2, pp. 84-86.

[2] F. Nedeoglo, A. Chepurnov, D. Komissarov, Linux and RT-Linux for accelerator control - pros and cons, application and positive experience. // Proc. of ICALEPCS'99, Trieste, Italy, ISBN: 88-87992-00-2, pp. 520-522.

[3] A.S. Chepurnov, F.N. Nedeoglo, D.V. Komissarov, Operating System Linux as Developing and Runtime Platform for Control System of Particle Accelerator, // Proc. of EPAC'2000

[4] SJA1000 Stand-alone CAN controller, Philipps Semiconductors, DATA SHEET, Preliminary Specification, 1997 Nov 04.

[5] A. Chepurnov, D. Komissarov, F. Nedeoglo, A. Nikolaev, "DeviceNet Implementations under Linux for Use in Control System of a Particle Accelerator." // Proc. of ICALEPCS'99, Trieste, Italy, ISBN: 88-87992-00-2, pp. 388-390.

[6] DeviceNet Specifications, Volume 1, Release 2.0, Volume 2, Release 2.0.

[7] V. Yodaiken, M. Barabanov, "RTLinux Version Two Design", // VJY Associates LLC, 1999, http://www.rtlinux.com/archive/design.pdf

[8] A.S.Chepurnov, A.A.Dorokhin, K.A.Gudkov, V.E.Mnuskin, A.V.Shumakov, "Family of Smart Devices on the base of DSP for Accelerator Control.", Proc. of ICALEPCS, W2B-d (Chicago, Illinois USA, 1995).

[9] CAN Specification Version 2.0, 1991, Robert Bosch GmbH, Postfach 50, D-7000 Stuttgard 1