Hard Real-Time Control of Mechatronic System: FPGA Extension for RTAI

S. Carabelli, M. Chiaberge, R. Garzella, L. Gobetto and A. Mazzetto
Laboratorio Interdipartimentale di Meccatronica - Politecnico di Torino
C.so Duca degli Abruzzi 24, 10124 Torino, Italy
stefano.carabelli@polito.it

Abstract

Hard Real-Time Control (HRTC) is to be intended in terms of the relative time scales of the electromechanical components of the mechatronic system to be controlled. MicroElectroMechanical Structures (MEMS) or Drive-by-Wire (DbW) applications are good examples of hard real-time controls.

The combination of algorithmic and logic processing power by means of micro and/or Digital Signal Processor (uP, DSP) and Programmable Logic Devices (PLD) or Field Programmable Gate Arrays (FPGA) results in a powerful Algorithmic and Logic Core (ALC) to be used in many different and demanding mechatronic applications.

The digital platform here proposed is an industrial PC equipped with a FPGA board whose firmware Intellectual Properties (IP) can be accessed by the different software Service Routines (SR). Firmware is memory mapped into the PC memory space and the interrupt management is handled by a combined structure of software scheduler and firmware interrupt manager.

The FPGA board is also used to implement a hard real-time bus to connect such a digital platform to a number of DSP/FPGA based Actuator Control Units (ACU) dedicated to current and torque fine drive and control.

The FPGA board driver is developed within the proprietary hard real time RTAI API's and the development tool chain can be fully handled within the Simulink environment: simulation, software and firmware code generation use the very same model.

1 Introduction

The world of mechatronics and robotics applications always requires new and innovative devices, which must be made available as soon as possible. From the first idea to the finished product it is needed a certain time period for good prototypation and engineerization processes.

To minimize the prototypation time, where the LIM (Interdepartimental Mechatronic Laboratory) use a lot of resources, they are needed tools whose key words be *modular structure* and *versatility.* Only using electronic devices or just software would be counteroproductive for the modelling and characterization of the problem, so at LIM it is from very much time that we use some suitable tools which combine excellent performances (like DSPs), reliability and programming easiness use. A characteristic feature is the ability to interface our systems with the rest of the world very easily, through whichever communication channel or bus, thanks to FPGA planned as the demand ask. The abstract vision (Fig. 1) that is come to create of our prototype is therefore characterized from the FPGA communicating with external devices and from DSPs/CPUs elaborating informations. The interaction with the user is managed again from DSPs/CPUs and it happens via HMIs (Human Machine Interfaces).

A condition “sine qua non”, absolutly needed, is the guarantee that the sampling and elaboration times are respected. In fact, a basilar concept is that the mechatronic discipline cannot prescind from the hard real time. In the prototypation devices adopted it is guaranted the hard real time through appropriate schedulers and software watchdogs.

The above abstract vision introduces an important feature: it is modular and it is possible to swap the elaboration unit with an other therefore like the hardware FPGA without changing the software or firmware. A complementary solution, not so hard real time operating system but still hard, it is really to use systems PC processors-based to the place of the DSP, or use them together giving to everybody certain tasks (we prefer service routines). This last solution it allows to relax the DSP and the CPU from very hardly work.

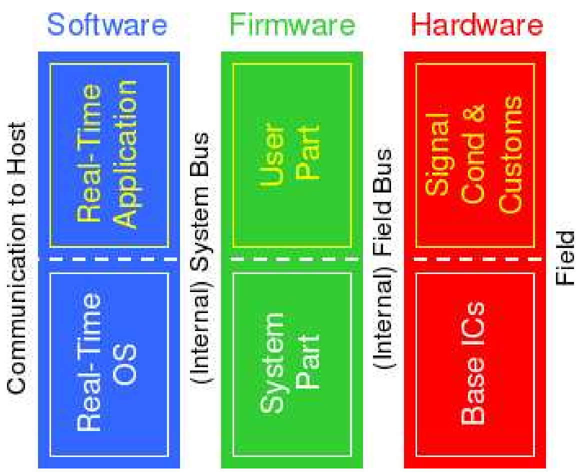


FIGURE 1: *System architecture*

Adopting standard CPU we ask a question, that is which operating system use to have the hard real time support. The LIM philosophy is near to the open source philosophy, in fact the Mechatronics Laboratory has coined ”open hardware” for its devices. From a careful analysis of the open source world wide solutions we have found in RTAI the best candidate.

Build the Linux/RTAI system (kernel recompilation) [2], we have thinked how to have a manageable programable logic in the PC. The step to found the solution has been much short, in fact we have adopted a PCI FPGA that we have used also for other plans. Equipped with drivers for just Microsoft Windows, we have written new Linux drivers optimized for RTAI.

The Linux/RTAI PC is a good start point for embedded devices turned to prototypation, that is small board that they integrate a PC processor like Intel XScale and one or more programable logic. The processor, whose architecture is usually RISC, would execute the Linux/RTAI code, while integrated programable logic the firmware of the FPGA before on PCI bus. With good ideas, will and a little of fortune, we have increased our knowledges and the solutions for a prototypation process faster and more efficient.

2 Open Digital Platform for Mechatronic Applications

At the core of almost any modern automotive, avionic and general industry product, there is some kind of programmable digital platform often called Electronic Control Units (ECUs). Such platforms are usually based on a combination of digital devices such as microProcessors (uP), micro Digital Signal Processors (uDSP) and/or custom Programmable Logic Device (PLD) in addition to application specific signal conditioning and power electronic circuitry.

The replacement of pneumatic or hydraulic actuators with electrical motors is definitely a general trend in the innovation of many products. This “more electric” technology shift is not limited to avionic or automotive industry but is a rather generalized evolution. Such technology will require dedicated ECUs, hereafter called Actuator Control Units (ACUs), capable of driving different kid of electric motors by mean of switched power electronics. ACUs must run their critical code under a deterministic real-time operating systems and must usually communicate by means of a dedicated network protocols.

Our ACU features a full set of digital programmable devices much alike those actually used in industrial applications: a uDSP device meant for electrical drives such as those of Texas Instruments C24/28 families, a Field Programmable Gate Array (FPGA) device (Fig. 2), and a Field Programmable Analog Array (FPAA) device. These devises allow to implement different task on the more adapt programmable real-time device. ACU features an additional Communication Module with standard protocols for copper wire and Plastic Optical Fiber (POF), a Power Module that can be used on any kind of electric motor including voice coil and electromagnets, and a number of Field Module to be used to implement application specific circuitry, e.g for sensor signal conditioning.

The ACU comes with a complete development tool chain that goes from graphical programming and automatic code generation in Matlab/Simulink enviornment, to local or remote virtual panels, code download and managing console, parameter tuners and signal scopes to be used for testing and maintenance.

The Open Source approach is here extended to include the whole system including the firmware and hardware as well as the software components. The open architecture of the proposed ACU allows manufacturers, system and component suppliers, academic researchers to modify and adapt the hardware, firmware, or software to their specific needs without depending on closed proprietary solutions and related services.

Replacing the DSP with a PC RTAI powered and the integrated FPGA with an FPGA on PCI bus, we have a complementary solution, with the same architecture and modularity. The PCI FPGA is a good idea to allow the user to communicate with the rest of the world, but it introduces a certain delay and some uncertainty when you want, for example, read from digital input lines. The PCI bus is not optimized for hard real time uses and its nature of interrupts shared can up-limit performances, so we prefer say “not so hard but still hard”. If the user has an embedded board RTAI powered with a dedicated and custom bus, it is possible to link a FPGA device (not PCI) and the above latency problem is resolved.

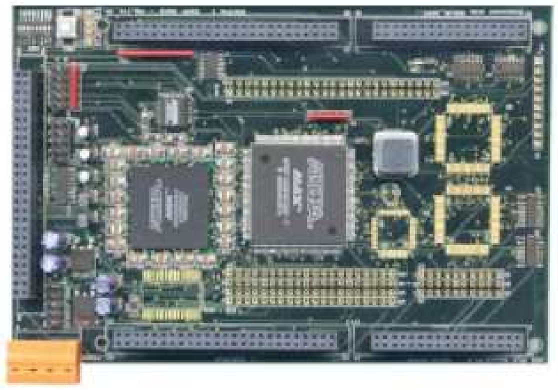


FIGURE 2: *Our board with FPGA feature*

2.1 ACU general architecture

The general architecture of the ACU system (with DSP or RTAI) is devised to allow the user to develop three different kind of technology components (Fig. 1):

- Software Service Routines (SR) to be excuted in Interrupt mode (ISR) or in Time Triggered mode (TTSR)
- Firmware Functional Blocks (FB) sometimes called Intellectual Properties (IP)
- Hardware Interface Circuitry (IC) to field application signals

In order to leave the user to concentrate on the design and testing of the above mentioned parts, an extended Operating System is to be provided together with the digital platform (an hard real time operating system, like our HRTOS on DSP or Linux/RTAI). An hard real time operating system extends the concept of operating system from the software component to the whole system operation. The user coded firmware functional blocks are to be managed by some other firmware functional blocks that take care of their interaction with the software on the DSP side and the hardware on the application field side.

The circuitry specific to every different application, e.g. sensor conditioning, can be thought as user defined hardware to be developed within a given framework of available analog and digital channels with their given electrical characteristic, i.e. something that allows the user to operate the system.

3 A Linux/RTAI driver for FPGA devices

The PCI FPGA card was supplied with just Microsoft Windows drivers, then we have written new for Linux operating system. But running a RTAI task that use the driver, you have continuously a hard real time/soft real time switches, with a meaningful worsening of performances. The only solution for this problem is to write a Linux driver using some RTAI APIs for kernel space.

Linux offers more than one possibility to write a driver [5], for example a character driver that create a file in */dev* directory accessible from a standard program with *fopen/fread/fwite/fclose* calls. In this case, when the program yelds the control to the driver, there is a context switching from user space to kernel space and viceversa after the appropriate function is ran. Also using RTAI APIs this happens but the context switching time is very insignificant. For our prototypes, we want the best, then our solution for the driver it has been various. In fact, we have preferred the pattern named *LXRT Extension* [3], that is we have created the driver (not character driver) and an header that it is possible to use in the user software to extend LXRT with our functions to access to the FPGA. This possibility seems not official documented but it is very efficent.

Our driver is a generic driver, in fact it allows to access directly to 32 bits registers simply calling a read or write function. So it is possible that an user, with its user space task, can custom the access to FPGA (he extends the driver). This is a classic feature very used at the Mechatronics Laboratory.

Then, loaded the driver as module, the user must just include an header in its service routine to use the FPGA and extend LXRT. Follow ANSI C main prototypes from header source code:

```
extern int rtplda_check(int);
extern unsigned int rtplda_read(unsigned int,_
   unsigned int, unsigned int, unsigned int *);
extern unsigned int rtplda_write(unsigned int,_
   unsigned int, unsigned int, unsigned int);

#ifndef DISABLE_IRQ
```

```
   extern unsigned int rtplda_mbxname(int,_
      char *);
#endif
```

The implementation of these functions is in the driver, but an user calls them as if they were implemented in the local program. When the user calls a function, like *rtplda_read(...)*, the RTAI core provides to call the related driver function. In this way, the context switching from user space to kernel space is minimized.

With our driver, it is possible to manage hardware interrupt too (if user decides to enable this feature, but why not?), simply listening on mail-box returned from *rtplda_mbxname(...)* function (the first parameter is the FPGA number, because our system manage up to 4 devices) that notify the FPGA has triggered an interrupt.

To test the driver we have done a funny experiment, so we have linked the FPGA with a function generator, then we have chosen a square wave and at each front an interrupt was triggered, then the user program sended a signal through a National Instruments DAQ PCI board. With an oscilloscipe, we have noticed that the gap between the input and output signal is very small because both our driver and the FPGA are very efficent.

The driver is released as free software under the GNU Geneal Public License Version 2 or above and was developed by our eclectic researcher Alberto Mazzetto.

4 Generating code for FPGA with Mathworks Symulink

Mathworks Symulink is the tool "de facto" for technical computing and in the RTAI package is present an extension to autogenerate ANSI C source code for Linux/RTAI based system. We have build new blocks (Fig. 3) to generate code for FPGA devices, so simply the user can place a block (i.e.: *RTAI/PLDA Write*) to interact with 32 bits registers, read and write is both supported. In this way, an user can create for example a powerful automatic control for its device and for us it is very easy to build a prototype time-cheaper.

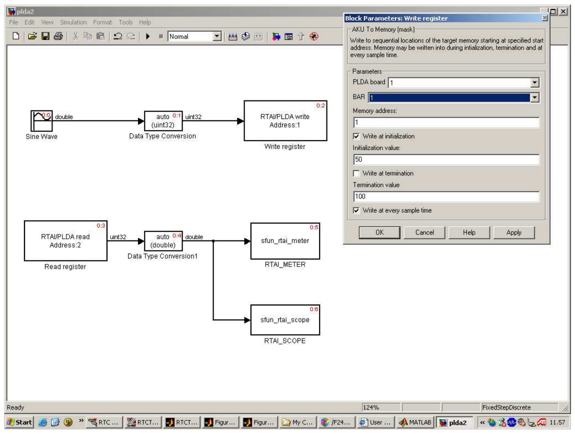
When in a Symulink project there is one or more FPGA blocks, the generation code tool add a link to *plda_rtai_lxrt.h* header (the same for user program manually wrote) and put in the source code lines like the following:

```
var=rtplda_read(..., ..., ..., ...);
var=rtplda_write(..., ..., ..., ...);
```

Obviously, it is possible to custom each block specifying each parameter that will be an argument for the relative function.

The Mathworks Symulink section of the binomial PC-FPGA was developed by Luigi Gobetto, a fundamental PhD researcher of the LIM.

A FPGA set blocks for Scilab/Scicos, an other designing environment to generate code, will be released as soon as possible.



Block Parameters: Write register

AKU To Memory (mask)

Write to sequential locations of the target memory starting at specified start address. Memory may be written into during intialization, termination and at every sample time.

Parameters

PLDA board: 1

BAR: 1

Memory address: 1

[✓] Write at initialization

Initialization value: 50

[ ] Write at termination

Termination value: 100

[✓] Write at every sample time

FIGURE 3: *Symulink and FPGA blocks*

5 A didactical application

A very interesting test rig have been build at the Interdepartimental Mechatronic Laboratory for public expositions as didactical demonstrators of magnetic levitation principle and applications (Fig. 4).

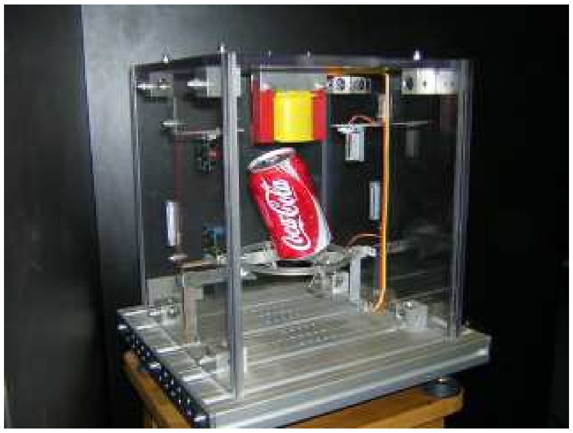


FIGURE 4: *The magnetic levitator*

Using the ACU digital platform, the electrical and position loops can be shown implemented as part of a DSP service routine and the FPGA can be used to perform specific conditioning on the levitator behaviour.

Introducing RTAI in our didactical experiment, the PC has become the control position manager with exactly the same code wrote for DSP (with Symulink) but adapted and recompiled for RTAI platform. In this case, the FPGA PCI board is used to communicate via digital protocol with the logic on ACU platform to coordinate each jobs.

6 Conclusion

An open and modular digital platform specially meant for mechatronic applications is shown to be conveniently based on a combination of DSP or PC-processor and FPGA digital devices. The availability of the describing or functional source code for every part of the system is also stressed as a major technical advantage for the development of complex mechatronic applications. RTAI is a good extension for Linux and its role in our system part make a complete and powerful digital platform.

References

[1] The RTAI Development Team, *DIAPM RTAI - Realtime Application Interface*, http://www.rtai.org.

[2] Rtai.dk, *Installation of RTAI-3.0r3 on Mandrake 9.2*, http://www.rtai.dk.

[3] Peter Soetens, *Extending LXRT*, http://people.mech.kuleuven.ac.be/∼psoetens/.

[4] Peter Soetens, *HOWTO Port your C++ GNU/Linux application to RTAI/Linux*, http://people.mech.kuleuven.ac.be/∼psoetens/

[5] Alessandro Rubini and Jonathan Corbet, 2001, *Linux Device Drivers*, O'Reilly, 2nd Edition.