THE NEW MULTICORE REAL-TIME CONTROL SYSTEM OF THE RFX-MOD EXPERIMENT

G. Manduchi, A. Luchetta, C. Taliercio, Consorzio RFX, Padova ITALY

Abstract

The real-time control system of RFX-mod nuclear fusion experiment has been in operation since 2004 to control the plasma position and the MagnetoHydroDinamic (MHD) modes. Over time new and more computing demanding control algorithms have been developed and the system has been pushed to its limits. Therefore a complete re-design was carried out in 2012. The new system adopts radically different solutions in Hardware, Operating System and Software management. The VME PowerPc CPUs communicating over Ethernet used in the former system have been replaced by a single multicore server. The VxWorks Operating System, previously used in the VME CPUs has now been replaced by Linux MRG that proved to behave very well in real-time applications. The previous framework for control and communication has been replaced by MARTe, a modern framework for real-time control gaining interest in the fusion community.

INTRODUCTION

Recent nuclear fusion experiments use digital real-time control to improve the quality of the plasma confinement. Plasma control is carried out by a combination of applied toroidal and poloidal magnetic fields. In addition, a vertical field is required to avoid that the plasma moves outwards and to shape the plasma column, and radial field components are used for the control of the plasma Magneto Hydrodinamic (MHD) instabilities. The magnetic field generated by coils driven by the control system extends the corrective action performed by the conducting shell surrounding the plasma container. Plasma instabilities, in fact, generate counterbalancing currents inside the shell. If on the one side this effect is immediate, and therefore much faster than what could be done by a digital control system, on the other one the corrective action soon terminates, due to the finite resistance of the shell and penetration time of the magnetic field. In practice, the shell alone is not able to fully compensate plasma instabilities which would soon lead to an increased interaction between the plasma and the container wall (and a consequent abrupt interruption of the discharge.

The actuators for the control system are the coils generating the corrective magnetic fields which can be grouped in:

- Toroidal coils generating the toroidal field component, i.e. along the torus;
- Field Shaping coils, generating a vertical field component;
- Saddle coils, located around the plasma container and generating a radial field

Even if magnetic control represents the most important part of the required actions, other actuators may be defined in fusion devices such as gas puffing for controlling the density of the plasma or additional heating to produce localized interaction with the plasma and to prevent disruptions.

The RFX-mod experiment[1] has 12 toroidal coils, 8 pairs of field shaping coils (connected in series) and 192 saddle coils, fully covering the torus in a 48x4 array, corresponding to 212 output signals generated by the control system. Toroidal and field shaping coils are used for axisymmetric control, that is, for generating corrective fields which are the same along the poloidal direction and therefore correcting global parameters such as the displacement of the plasma column. The saddle coils are used to provide localized corrections in order to control MHD instabilities. The input signals of the control system are derived from a set of electromagnetic probes located in several positions around the torus. In particular, corresponding to every saddle coil position, the radial and toroidal component of the electromagnetic field are acquired as well as the actual value of the current flowing in the coil. It is worth noting that the physical quantities used for control computation, such plasma position and current, are not directly derived by measurements, but they are derived by pre-processing the signals acquired by electromagnetic probes. The control system of RFX-mod acquires 192x3 signals taken at the position of the saddle coil actuators and another set of 128 signals coming from a variety of electromagnetic probes located in different places.

In addition to the number of input and output signals, another important factor for the characterization of a control system is the maximum allowable latency. As stated before, the first reaction to plasma instability is provided by the conducting shell around the torus controlling in this way those phenomena which are too fast to be handled by a control system. The shell counterbalancing action, however, terminates after a few tens of microseconds, and the intervention of the control system is then needed. For this reason the maximum latency of the control system is in the range 50-100µs for the fastest phenomena, such as vertical instabilities. Other physical quantities have a slower dynamics, especially in large devices, requiring a minimum latency of 1-10 ms.

Two other important factors must be considered in the definition of the architecture of digital control systems for fusion devices: determinism in response time and reliability. Both factors may heavily affect the development cost, but, luckily, they are not as stringent as they might be in other critical applications such as space missions.

ISBN 978-3-95450-139-7

As for deterministic timeliness, it is to be noted that control systems in fusion devices are typically used for feedback control. An occasional miss in time deadline produces therefore a missed update of the output reference for one cycle. Provided that the used control algorithm has a stability margin large enough, this event does not affect control in practice.

As for reliability, the typical approach in the implementation of fusion research devices is to define a separate interlock system for handling all those events which may affect the integrity of the machine, including control system failures. The interlock system is much simpler than the control system and therefore full reliability can be guaranteed for the interlock system. Providing full reliability in the control system would complicate quite a lot its implementation and consequently increase its cost. As an example, ITER, the larges fusion device currently under construction, defined three separate protection tiers [2]: Safety, Interlock and Control. The safety system ensures the most critical protection, involving nuclear risk management and personnel protection. The interlock system is involved in all those events which may affect the integrity of the machine, and its failure represents an event to be handled by the safety system. The control system is responsible of the complex management of the machine operation and its failure represents an event handled by the interlock system.

MIGRATION FROM THE PREVIOUS RFX-MOD CONTROL ARCHITECTURE

The first digital control providing both axisymmetric and MHD controls was implemented at RFX-mod in 2004 and consisted in a network of 7 VME chassis, each hosting a Motorola MVME 5500 single board computer and Analog to Digital (ADC) or Digital to Analog (DAC) converters [3]. Some chassis carried out data acquisition and pre-elaboration, while the others provided the computation of the control algorithms and the generation of the output reference waveforms for the power supply system feeding the actuator coils. Communication among the system components was carried out by an insulated 1Gbit/s Ethernet segment, and the single board computers were running the VxWorks Operating System (OS).

Some design choices made in that system proves successful during the system lifetime, in particular:

- The use of a general purpose solution (Ethernet) for communication in place of more specialized ones, such as reflective memories, allowed an easy update of the system in 2006 by replacing the former 100Mbit/s Ethernet with 1Gbit/s Ethernet;
- The definition of a modular and distributed architecture spreading the computational load among different machines. New controls have been easily integrated during the system lifetime by adding more nodes in the network.

Other design choices proved right but information technology provides nowadays better solutions. For example, the use of VxWorks represented a mandatory choice because at that time Linux could not be considered a real-time OS. In recent years, however, the Linux kernel has been greatly improved in terms of latency by making the kernel core pre-emptible, and recent extensions provide a real-time performance similar to that of VxWorks [4].

During the system lifetime, new and more complex control algorithms have been integrated, and the performance of the control system increased until hitting its intrinsic limits, in particular computing power and latency.

In 2012 the system was replaced by a new one which, if on the one side it retains the modular and distributed nature of the old system, on the other one represents a radical change both in hardware and software [5].

The network of VME single board computers has now been replaced by a single multicore Linux server by mapping the functions carried out by the single board computers onto the cores of the server. Multicore processors represent nowadays a widespread solution not only in servers but also in laptops and even in smartphones. Increasing the number of replicated cored into processors does not always imply a proportional increase in performance, as performance depends also on the degree of parallelization in the used algorithms. In our case, however, control was already partitioned among different communicating components and mapping the previous system into the new one has been a straightforward task by assigning the tasks carried out by every former VME single board computer to the cores of the server in a one-to-one relationship. 11 out of 12 cores available in the server (HP ProLiant DL 370G6) have been used and they have partitioned in three sets: pre-processing, control and output generation (Fig. 1). The four cores user for pre-processing supervise data acquisition for the different sets of signals and carry out the required pre-processing in order to generate the physical quantities which are then used in control. Data can be acquired in two ways:

- Direct data acquisition via PCIe-based bus extenders. In this case the ADCs are mounted on separate chassis (cPCI, PXI or ATCA) which are connected to the server via a bus extender, using PCIe for communication. In this configuration the device registers are mapped into the I/O address space of the server and high speed in data transfer can be achieved. It is worth noting that large data throughputs are not required in control system. For example, acquiring 100 ADC channels using 16 bit for the conversion at a rate of 10 kSample/s implies a data throughput of 2 MB/s which represents an almost negligible amount when compared to the data throughput normally achieved in disk accesses.
- Network data acquisition. This solution has been temporarily adopted in RFX-mod due to budget

limitations which forced the reuse of the VME ADCs. In this case raw data are acquired by the VME controller CPU and sent via UDP packets. The replacement of the VME ADCs with new ATCA ADCs, directly connected to the server via a bus extender, is foreseen in 2014, thus removing the latency due to network communication and IP stack management.

Three other cores are used to carry out the thee control algorithms for the generation of the reference waveforms to the power supply units of the saddle, field-shaping and toroidal coils, taking pre-processed data from shared memory and producing the reference values. The remaining set supervises DAC generation, and it would not have been required if the DAC device drivers (National Instrument PXI NI6723 PXI) had supported DMA data transfer. Unluckily, this is not the case and the time spent for performing direct I/O transfer to the DAC devices would have reduced the time available for control computation if data generation were carried out by the same cores performing MHD, toroidal and axisymmetric control.

Communication among cores is carried out via shared memory. As all the involved threads belong to the same process, they share the same address space and therefore communication does not require the use of specific system calls for shared memory management. POSIX semaphores and mutexes have been used to protect data integrity and to achieve thread synchronization.

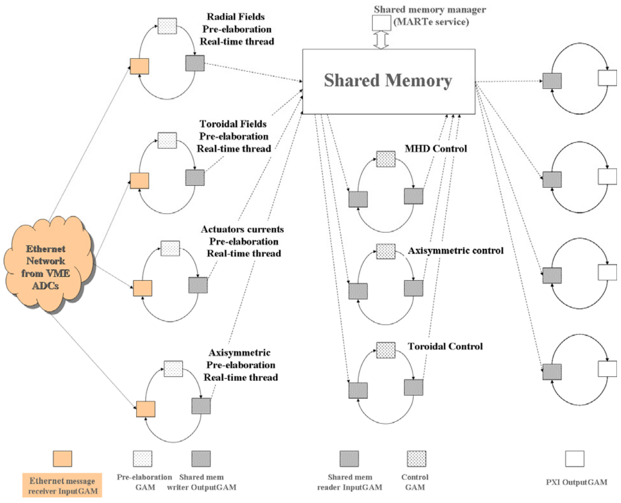

Figure 1. Real-time control threads organization

SYSTEM CONFIGURATION

A fixed core assignment has been possible thanks to the fact that there are enough available cores to assign a single control thread to cores. 11 out of 12 cores are dedicated to the control system and the left core is used for the normal system activity. Fixed core assignment is carried out by:

- Using the ISOLCPU Linux boot parameter to instruct the Linux scheduler to use only the core not used in control for task assignment;
- Explicitly assigning threads to cores in the framework code using the cpu_setaffinity() system call. In this case the created thread is assigned to the core specified in the passed mask regardless the ISOLCPU definition.

The combined usage of ISOLCPU and sched_setaffinity() removes most jitter in latency due to the non determinism in thread core assignment which would happen if the Linux scheduler were allowed to use all the cores for thread execution. A static assignment of cores to thread is mandatory to reduce jitter in system latency. However, assigning a single thread to each core represents somehow a fortunate situation where the number of available cores is enough. For larger systems the number of threads may exceed the number of available cores and in this case it is necessary to handle the co-existence of multiple threads in the same core. Two approaches are possible:

- Define the processor mask passed to sched_setaffinity() so that more than one thread is assigned to a given core. In this case freedom of the Linux scheduler is still limited, as it can assign only a given core to each threads, but the scheduler will now handle the required context switches among the threads assigned to the core;
- Enable processor hyperthreading in the BIOS configuration in order to double the number of cores seen by the OS. When hyperthreading is enabled, the system takes advantage of available replicated components in cores, but it does not carry out a full duplication of the hardware resources which are contended by different threads, even if running on different virtual cores.

It is not easy to state in principle which solution is preferable, so we carried out a test where only half of the available physical cores were used. It turned out that hyperthreading presents a significantly degraded latency in respect of the other, probably due to a less flexible assignment of the contended resources.

LINUX IN REAL-TIME APPLICATIONS

At the time the original real-time control system has been built, the usage of VxWorks as real-time OS for critical applications was almost mandatory, even if since then Linux was rapidly increasing its user base. Things changed in the following years with the advent of version 2.6 introducing kernel preemption and therefore improving real-time responsiveness. A further step in this direction is represented by the PREEMPT_RT patch of the Linux MRG distribution [6]. Linux MRG provides preemptible critical sections, priority inheritance for kernel semaphores and preemtible interrupt handlers. While in kernel 2.6 the mechanism used to protect critical section in the kernel are implemented as spinlocks, in PREEMPT_RT it is implemented by a new version of semaphore (called rt-semaphore) whose implementation is efficient if the underlying architecture supports compare & exchange instruction. Priority inheritance provides protection against priority inversion thus reducing the likelihood of delays introduced by lower priority tasks. Preemptible interrupt handlers are implemented by leaving the minimum amount of core in the Interrupt Service Routine (ISR) and moving the rest of the required driver actions into real-time threads which are managed by the Linux scheduler based on their priority. As a result, the real-time responsiveness of Linux MRG is improved for important (i.e. with high priority) tasks and the jitter in latency is reduced.

SOFTWARE FRAMEWORK

The radical change in architecture carried out in the new system includes also the software framework. Instead of refactoring the previous software framework, we decided to adopt a solution shared with other laboratories and we selected the MARTe framework, initially developed at JET and currently adopted in different fusion laboratories. MARTe provides the following features:

- It is oriented towards multicore architectures defining a number of threads within the context of a single Linux process. Every thread is configurable including its core assignment
- It is modular: its configuration is defined in a text file which is parsed at system startup. The configuration file defines, among others, the number of threads, their core assignment and the operations carried out by every thread. A change in configuration requires only a change in the configuration file, not in the code.
- It is agnostic on the kind of computation being carried out. What the framework provides is the required mechanisms for data flow and the supervision of the running threads. Actual computation is deferred to user-provided components called Generic Application Modules (GAMs). These modules are implemented as C++ classes inheriting from a generic GAM superclass which defines the standard interface of MARTe components.

The configuration defined in the configuration file can be easily replicated in different systems. This feature turned out very useful for simulating the control system by replacing in the configuration the components performing data I/O with others reading and writing offline data. To use the MARTe framework it is therefore necessary to develop the system-specific components. However, the developer in now only concerned on his/her component, exchanging data with the rest of the system via a simple interface, thus hiding the complexity of the underlying system.

REFERENCES

[1] R. Piovan et al. “RFX machine and power supply improvements for RFP advanced studies”, Fusion Engineering and Design vol 56-57, pp. 819-824 October 2001.

[2] J.B. Lister, J.W. Farthing, M. Greenwald, I. Yonekawa “The status of the ITER CODAC conceptual design” , Fusion Engineering and Design vol 83, pp. 164-169 April 2008.

[3] M. Cavinato, G. Manduchi, A. Luchetta, C. Taliercio “General Purpose Framework for Real Time Control in Nuclear Fusion experiments” IEEE Transactions on Nuclear Science vol 53, pp. 1002-1007 June 2006.

[4] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, C. Taliercio “Performance Comparison of VxWorks, Linux, RTAI, and Xenomai in a Hard Real-Time Application” IEEE Transactions on Nuclear Science vol 55, pp. 435-438, February 2008.

[5] G. Manduchi, A. Barbalace, A. Luchetta, A. Soppelsa, C. Taliercio, E. Zampiva “Upgrade of the RFX-mod real time control system” Fusion Engineering and Design, Vol. 87, pp. 1907-1911, December 2012.

[6] CONFIG PREEMPT RT Patch Home Page, https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch