Fusion Engineering and Design xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Continuous data recording on fast real-time systems

L. Zabeo[a,*], F. Sartori[a], A. Neto[b], F. Piccolo[a], D. Alves[b], R. Vitelli[c], A. Barbalace[d], G. De Tommasi[e], JET-EFDA contributors, See the Appendix of F. Romanelli, et al., Proceedings 22nd IAEA Fusion Energy Conference, Geneva, Switzerland, 2008.

[a] Euratom-CCFE, Culham Science Centre, Abingdon, Oxon OX14 3DB, United Kingdom
[b] Associação Euratom-IST, Instituto de Plasmas e Fusão Nuclear, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
[c] Dipartimento di Informatica, Sistemi e Produzione, Universitá di Roma, Tor Vergata, Via del Politecnico, 1-00133 Roma, Italy
[d] Euratom-ENEA Association, Consorzio RFX, 35127 Padova, Italy
[e] Associazione EURATOM/ENEA/CREATE, Universitá di Napoli Federico II, Napoli, Italy

ARTICLE INFO

Article history:
Received 12 June 2009
Received in revised form 2 February 2010
Accepted 3 February 2010
Available online xxx

Keywords:
Real-time
RTAI
Data streaming

ABSTRACT

The PCU-Project [1] launched for the enhancement of the vertical stabilisation system at JET required the design of a new real-time control system with the challenging specifications of 2 Gops and a cycle time of 50 μs. The RTAI based architecture running on an x86 multi-core processor technology demonstrated to be the best platform for meeting the high requirements. Moreover, on this architecture thanks to the smart allocation of the interrupts it was possible to demonstrate simultaneous data streaming at 50 MBs on Ethernet while handling a real-time 100 kHz interrupt source with a maximum jitter of just 3 μs.

Because of the memory limitation imposed by 32 bit version Linux running in kernel mode, the RTAI-based new controller allows a maximum practical data storage of 800 MB per pulse. While this amount of data can be accepted for JET normal operation it posed some limitations in the debugging and commissioning of the system. In order to increase the capability of the data acquisition of the system we have designed a mechanism that allows continuous full bandwidth (56 MB/s) data streaming from the real-time task (running in kernel mode) to either a data collector (running in user mode) or an external data acquisition server. The exploited architecture involves a peer to peer mechanisms where the sender running in RTAI kernel mode broadcasts large chunks of data using UDP packets, implemented using the 'fcomm' RTAI extension [2], to a receiver that will store the data. The paper will present the results of the initial RTAI operating system tests, the design of the streaming architecture and the first experimental results.

1. Introduction

The new vertical stabilisation system recently developed at JET has required the design of a real-time control system able to run at a cycle time of 50 μs with a very low jitter (¡2 μs). This challenge drove the development of a new high performance framework where to run real-time applications. The main objectives of the architecture were not only oriented towards a fast system but also to develop a flexible, easy to test and debug and exportable environment.

High performance leads to the requirement of managing a large amount of data generated by the real-time application. Because of the limitation of the local memory available an advance and efficient mechanism of data streaming had to be designed. As a major requirement, the mechanism must provide high transfer rate without interfere with the hard real-time tasks.

In the following section the real-time framework will be presented. The basic concepts of the implementation and the peculiarity of the system will be introduced.

In Section 3 the streaming mechanism is described followed by the test results.

In the last section the conclusion and the additional possible use of the framework will be treated.

2. Real-time framework

MARTe [3] (Multiplatform Application Real-Time executor) is an highly flexible and powerful framework developed at JET for implementing real-time control systems. The framework is based on a multiplatform C++ library named BaseLib2 (the library is already ported for VxWorks®, Linux®, Linux-RTAI®, Solaris® and MS Windows®) that optimises all the low level operating sys-

* Corresponding author. Tel.: +33 44 217 65 24; fax: +33 44 225 63 66.
E-mail address: lzabeo@jet.uk (L. Zabeo).

0920-3796/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.fusengdes.2010.02.006

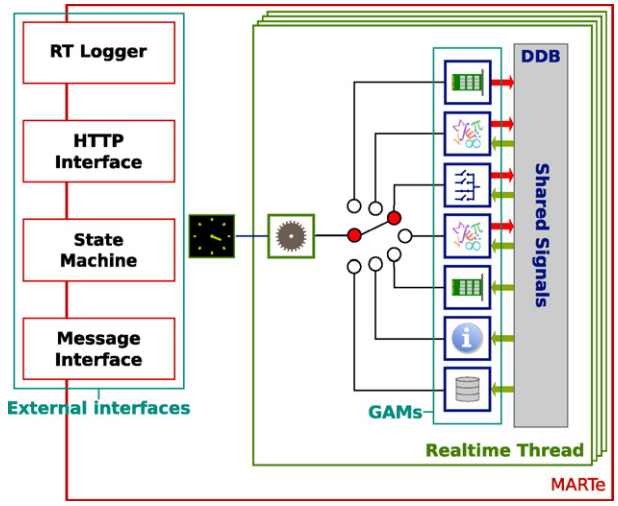

Fig. 1. MARTe components. The picture shows the list of GAMs and their connections with the DDB common database. MARTe operates as scheduler and implements all the external communications.

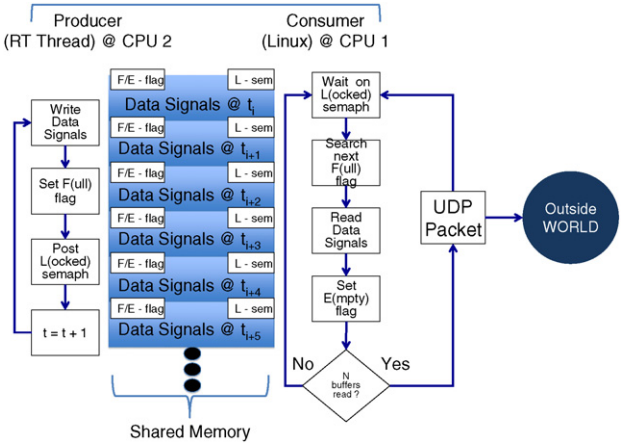

Fig. 2. UDP streaming architecture. For each cycle of the real-time thread a shared buffer is filled with the latest signal samples. When a pre-defined number of full buffers is reached an UDP packet is sent by another thread, providing a complete decouple of the two mechanisms.

tem functionalities and implements, on a layering structure, an object oriented organisation. Objects can be created and initialised automatically and registered in a common internal database to be available for the application. In particular the library includes a communication mechanism that allows messaging between objects, a powerful debugging mechanism that enables to retrieve all the information concerning an object and a flexible monitoring functionality that allows navigating the objects and visualise information via standard HTTP.

The real-time framework MARTe is exploiting the functionality offered by BaseLib2 by implementing a mechanism of scheduling that runs in a sequence for each cycle time a set of elaboration modules. Each modules named GAM (Generic Application Module) is able to perform from very simple to more complicated operations on a set of input data and producing a collection of output data (Fig. 1).

GAMs are not directly connected together but via a common signal database called DDB (Dynamic Data Buffer) where they can exchange data. The signals are identified by unique names and declared to the database in the initialisation phase of each GAM.

The mechanism allows flexible development of an application simply organising the sequence of GAMs in a way to perform a specific set of operations. It is quite simple to build or modify an application just by replacing, adding or re-ordering the set of modules. The input/output modules like the device drivers for the I/O cards (i.e. ADC, DAC communication modules) are implemented as GAMs (IOGAMs) as well.

The execution list of GAMs is passed to MARTe via a configuration file based on BaseLib2 standard language. MARTe not only orchestrates the execution but also provides the communication interface with the external world (HTTP interface, real-time logger, state machine for running the applications like for JET pulses and messaging interface). Moreover, MARTe can control multiple real-time threads where to run different collection of GAMs acting as supervisor.

The modular approach of the framework improves the efficiency of the debug and test phases allowing to focus on a single GAM at the time without the interference of the other modules. Moreover the chosen solution allows simulation of real-time environment in an offline system (even running on a different platform) simply replacing the real-time input with a different input source.

The MARTe architecture is clearly oriented to the multi-core technology where the optimal performance can be reached. Running MARTe supervisor on one core and leaving the real-time threads on the remained cores is the optimal environment.

In order to reach the high performance required by the Vertical Stabilisation the Linux-RTAI on a x86 multi-core processor technology has been identified as the best solution compared to the tested VxWorks® based machines [2]. The hard real-time thread are running on RTAI kernel mode allowing high level of performance where the supervisor can be allocated in Linux-user mode where the same level of performance is not required. RTAI is responsible for the real-time task scheduling, inter-process communication mechanism management (semaphore, shared memory etc.), memory control and of scheduling Linux to execute whenever there is time available. RTAI provides a set of services which allow interrupts to be served. This has been possible intercepting all the interrupts and propagating all the non real-time interrupts to Linux. This integrated solution permits the real-time threads to run without interference from what is happening on the others cores.

3. Stream

Data streaming is provided by a high level driver, driven by a generic IOGAM. It can be configured either to stream data out, still guaranteeing the real-time execution of the remaining modules, or to serve as a data supplier to a MARTe system. The underlying protocol for data streaming is based on the User Datagram Protocol (UDP) implemented using the 'fcomm' RTAI extension [2].

The output streaming driver is divided in 3 sections: (i) an entry point function, called by MARTe to write the DDB signals that are to be sent; (ii) a parallel thread providing the streaming capability; (iii) a segmented shared memory enabling the connection between these two parts. Each memory segment contains an header with the current time in microseconds, followed by a block with enough size required to store one sample of each signal. For each loop of the real-time thread, the next free memory block will be filled with data and marked as full. Synchronization between consumer and producer is achieved using a collection of atomic memory locks. This scheme is depicted in Fig. 2 where the design is being used to stream between to different cores in the same CPU.

The number of data blocks defined to be streamed in each UDP packet will determine the output bandwidth of the system, given by this value times the number of signals to be written. The amount of segmented blocks that can coexist in the shared memory must be equal or greater than the defined number of data blocks to stream in a UDP packet.

Please cite this article in press as: L. Zabeo, et al., Continuous data recording on fast real-time systems, Fusion Eng. Des. (2010), doi:10.1016/j.fusengdes.2010.02.006

In order to guarantee the real-time sustenance, the data writer will not wait for a data block to be consumed by the streamer thread and will overwrite any data which has not been sent only, issuing a message warning that the system might be under dimensioned. The only limitation on the quantity of blocks in the shared memory area is the amount of memory available on the system. As so the quantity of shared data compartments can in principle be made very large, allowing the system to guarantee both real-time and data availability.

Ideally the consumer thread should live in a different core in order to avoid spurious and highly undesired interruptions of the real-time loop. If it is not possible to physically separate the two threads, it must be set with a priority lower in respect to the main executor. This design choice imposes the real-time guarantee to prevail over data availability.

3.1. Vertical stabilisation example

In the case of the JET vertical stabilisation (VS) a MARTe system, running in RTAI kernel mode, is responsible for the execution of a set of 18 GAMs in less than 50 μs. Unfortunately the RTAI kernel mode has a memory limitation of 1 GB, constraining the amount of data that can be stored in the local memory during an experiment. In fact the present version running at JET is able collecting around 200 signals at 20 kHz for approximately up to 45 s of continuous data recording.

The vertical stabilisation controller hardware is based on an Intel Quad-Core processor. Benefiting from the multi-core technology two completely independent MARTe systems were able to be deployed. The first, living in a single core and running Linux is responsible for data storing, while the second, running in RTAI kernel space, contains the control entities referred above but with the data storage GAMs removed. Both systems are connected using the streaming technology together with the local network interface of the controller.

4. Tests and performance

A set of benchmarking tests were performed in order to assess the maximum achievable data throughput. The first objective was to guarantee that the real-time was still conserved on the data producer. The VS MARTe control system was used as the data producer; as so a control loop cycle of 50 μs with a jitter inferior to 1 μs had to be guaranteed. In the receiver side, a standard Linux box was setup. Both machines were connected through a gigabit Ethernet switch.

The amount of data sent in each UDP packet is given by the number of signals (multiplied by the size of float) requested to be copied by the real-time thread, times the number of buffers to be transferred. As there is a latency associated with each UDP send, the size of the packet should be maximized. The maximum UDP value in the Linux kernel 2.6 series is set by default to 65,536 bytes.

The first test to be performed was related to the size of the UDP packet and its impact on the bandwidth. The packet size was increased by adding more signals to be copied in each loop cycle by the real-time thread. The number of transferring buffers was fixed in 10, so that at each 500 μs a packet was ready to be sent. As reported in Table 1 this test has shown that the limit for the bandwidth is 56 MB/s. For larger packets the system could not assert the real-time performance anymore. The limit has been identified to be due to the large amount of data (700 signal samples) that had to be copied into memory at each 50 μs control cycle and the associated time latencies.

In order to assert the results provided by the latest test, a second assessment was performed increasing the bandwidth by accumulating the number of signals while keeping the maximum UDP packet size set to approximately its maximum valuable. In order to do so, the number of buffers to be transferred in each packet was changed at each evaluation. The results, reported in Table 2, show that the same bandwidth as the first test has been reached confirming the limit of the system.

Table 1
Bandwidth assessment. The maximum bandwidth is limited by the number of signals to copy and by the highly demanding loop cycle of 50 μs. Number of buffers has been set to the value of 10.

N Signals	Bandwidth (MB/s)
50	4.08 ± 0.04
105	8.48 ± 0.08
205	16.48 ± 0.14
410	32.88 ± 0.43
615	49.28 ± 1.36
700	56.08 ± 1.10

Table 2
Effect on the achievable bandwidth as a function of the number of samples and the number of buffers to copy. These values corroborate the results achieved in the previous test.

N Signals	*N* Buffers	Bandwidth (MB/s)
50	310	4.08 ± 0.01
105	150	8.47 ± 0.02
205	77	16.48 ± 0.06
410	38	32.88 ± 0.33
615	25	49.28 ± 0.47
700	19	56.08 ± 0.75

In order to evaluate the performance and the quality of the system an header with a counter has been attached to each sent packet. The combination of the number of packets received in a time interval obtained using the accurate internal timer of the receiver allowed to determine the bandwidth of the system. The information has also been used to evaluate the correctness of the transmission demonstrating no data loss.

5. Conclusions and future developments

The new real-time multiplatform framework developed at JET has demonstrated to be an optimal environment for developing applications at high performance. The modular approach adds flexibility and reusability together with an improved debugging and testing capabilities without compromising the real-time performance.

The optimal results are reached with the multi-core processor technology where each hard real-time task and the less demanding supervisor (MARTe) are allocated on a different CPUs. The Linux-RTAI has shown to be the best platform thanks to a wise usage of the interrupts management.

In order to cope with the huge flow of data coming from real-time applications, a new streaming mechanism has been developed for continuously transfer data from the hard real-time task to a collection target. An impressive 56 MB/s has been reached on a standard Gigabit network without compromising a 50 us cycle with jitter in the order of the hundreds of nanoseconds.

The developed solution can be easily exported in an environment where the duration of the experiment doesn't allow local data storing like ITER. Continuous sending of data during a long pulse can be solved with the new data recording system.

Moreover with this architecture it could be possible to send a collection of data from the real-time tasks to offline post-processing systems. A chain of post elaboration can be attached to the real-time application providing elaborated data or visualisation without interfering with the experiment nor having to wait for the end of the experiment.

Please cite this article in press as: L. Zabeo, et al., Continuous data recording on fast real-time systems, Fusion Eng. Des. (2010), doi:10.1016/j.fusengdes.2010.02.006

Acknowledgements

This work was jointly funded by the United Kingdom Engineering and Physical Sciences Research Council and by the European Communities under the contract of Association between EURATOM and CCFE. The views and opinions expressed herein do not necessarily reflect those of the European Commission. This work was carried out within the framework of the European Fusion Development Agreement.

References

[1] F. Sartori, F. Crisanti, R. Albanese, G. Ambrosino, V. Toigo, J. Hay, et al., The JET PCU project: An international plasma control project, in: Proceedings 6th IAEA-TM, Inuyama, Japan, 2007.
[2] A. Neto, F. Sartori, F. Piccolo, A. Barbalace, R. Vitelli, H. Fernandes, et al., Linux real-time framework for fusion devices, in: Proceedings 25th SOFT Conference, Rostock, Germany, 2008.
[3] A. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, et al., MARTe: a Multi-platform Real-time Framework, in: Proceedings 25th IEEE Conference, Bejigin, China, 2008.

Please cite this article in press as: L. Zabeo, et al., Continuous data recording on fast real-time systems, Fusion Eng. Des. (2010), doi:10.1016/j.fusengdes.2010.02.006