Fusion Engineering and Design 85 (2010) 374–377

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[1]

[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:
Available online 1 March 2010

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).

[1] See the Appendix of F. Romanelli, et al., Proceedings 22nd IAEA Fusion Energy Conference, Geneva, Switzerland, 2008.

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

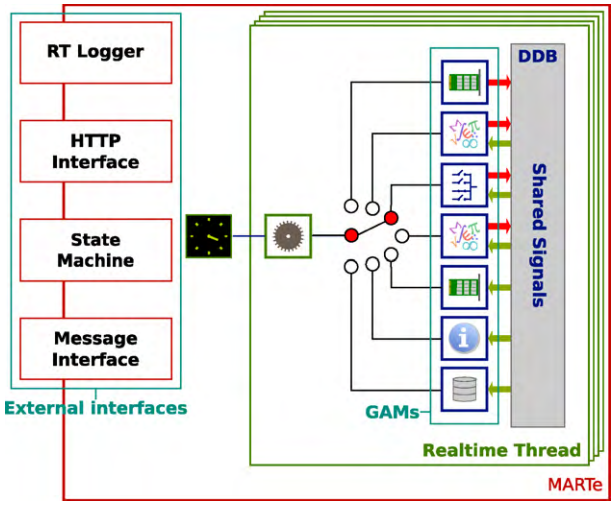

Fig. 1. MARTe components. The picture shows the list of GAMs and their connections with the DDB common database. MARTe operates as scheduler and implements all the external communications.

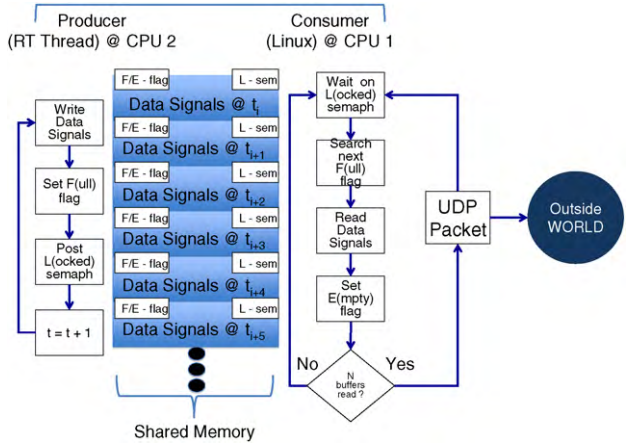

Fig. 2. UDP streaming architecture. For each cycle of the real-time thread a shared buffer is filled with the latest signal samples. When a pre-defined number of full buffers is reached an UDP packet is sent by another thread, providing a complete decouple of the two mechanisms.

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.