EMBEDDED EPICS ON µITRON/SH4-BASED CONTROLLERS

G. Jiang, J. Odagiri, N. Yamamoto, A. Akiyama, K. Furukawa, T. Katoh
High Energy Accelerator Research Organization (KEK), Tsukuba, 305-0801, Japan

ABSTRACT

Many Japanese Ethernet-based controllers being used in accelerator control systems adopt µITRON for their real-time kernels. Since those controllers have enough CPU power and memory capacities, IOC core program (iocCore) can run directly on them to realize "Embedded EPICS". The only viable option to achieve Embedded EPICS on various different controller types is to utilize BSPs available on the market as they are, since developing BSPs every time we support a new target is too expensive. Based on preceding investigation on technical feasibility, we have ported iocCore onto a target running µITRON on an SH4 CPU [1]. As a result, a fully functional IOC, which has all of the IOC software components such as Channel Access (CA) server and run-time database, was realized on the µITRON/SH4-based target.

In addition, we carried out a jitter measurement to evaluate the real-time performance of the µITRON-based IOC. The result showed that, after fixing a problematic code of EPICS for time difference calculation, the µITRON-based IOC has the real-time responsiveness as expected.

INTRODUCTION

Traditionally, various field-busses (CAMMAC serial highway, GPIB, Serial, CAN-bus, Profi-bus, MIL1553, etc.) have been used in accelerator control systems. On the other hand, modern accelerator control systems use more and more intelligent device controllers with an Ethernet interface. As a matter of fact, new accelerator projects such as J-PARC [2] and RIBF [3] are going to adopt Ethernet-based controllers listed in Table 1. This kind of Ethernet controllers can be directly connected onto the control network to replace traditional field-busses with Ethernet.

Controller	**Supplier**	**Kernel**	**CPU**	**RAM (min)**
MCU	Nichizo	µITRON	SH4	64MB
e-RT3	Yokogawa	µITRON	SH4	32MB
EMB-LAN100	Custom	µITRON	SH3	8MB
N-DIM	Custom	µITRON	SH4	6MB

Table 1: Characteristics of some typical Ethernet-based controllers

In Table 1, MCU and e-RT3 are commercial products while EMB-LAN100 [4] and N-DIM are custom ones. EMB-LAN100 is developed by KEK for the control of power supplies of DTL Q-magnets of J-PARC accelerator complex. N-DIM is developed by RIKEN for general-purpose control and monitoring required for the operation of RIBF. The left side of Fig 1 shows how those Ethernet-based controllers are being used in an EPICS-based control system.

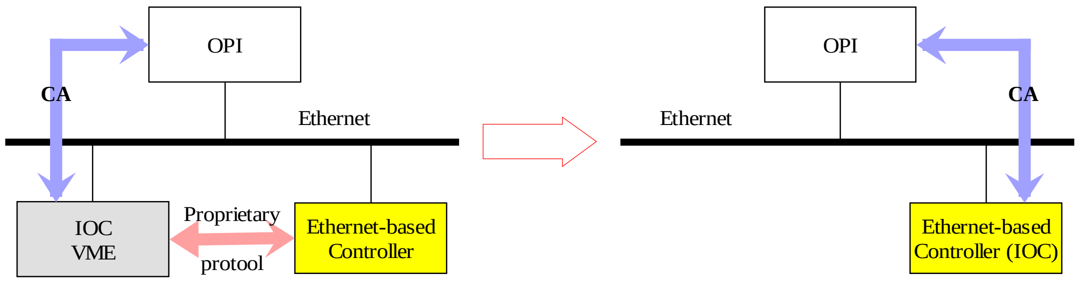

Fig 1. Structure of exclusive EPICS controller and Embedded EPICS controller

If we imagine the case where only Ethernet-based controllers are used in the system as an extreme case of Ethernet-based control system, expensive VME computers will end up just a protocol converter from proprietary protocol of each device to EPICS CA protocol, leaving all VME slots unused. One way to improve this inefficient use of the hardware resource is to replace the VME computer with an inexpensive computer such as PC104 running Linux. Another way, which is more sophisticated, is to run iocCore on each Ethernet-based controllers, as shown in the right side of Fig1, to make them Embedded EPICS controllers.

EMBEDDED EPICS ON µITRON

There are several candidates for the Operating System (OS) to realize Embedded EPICS since EPICS base 3.14 supports many OSes [5]. In fact, solutions based on Embedded EPICS have been around on processor cards running VxWorks, Linux or RTEMS [6, 7]. However, in general, Embedded EPICS tends to be a costly solution because development of BSPs is a time-consuming process. The cost is inevitable if the target hardware is custom made, and arises every time we support a new target.

On the other hand, in many cases, BSPs come with the hardware if the target is a commercial product. In the field of Japanese embedded application, µITRON is a very popular real-time kernel [8]. As shown in the Table 1, all of the devices listed use µITRON. Hence we can run EPICS on them without implementing BSPs on our own once OSD, which interfaces EPICS with µITRON, has been implemented. Creating a thin layer of device support on top of a BSP will suffice to support a new target in this approach. We decided to port EPICS onto a µITRON target as the first step toward this direction.

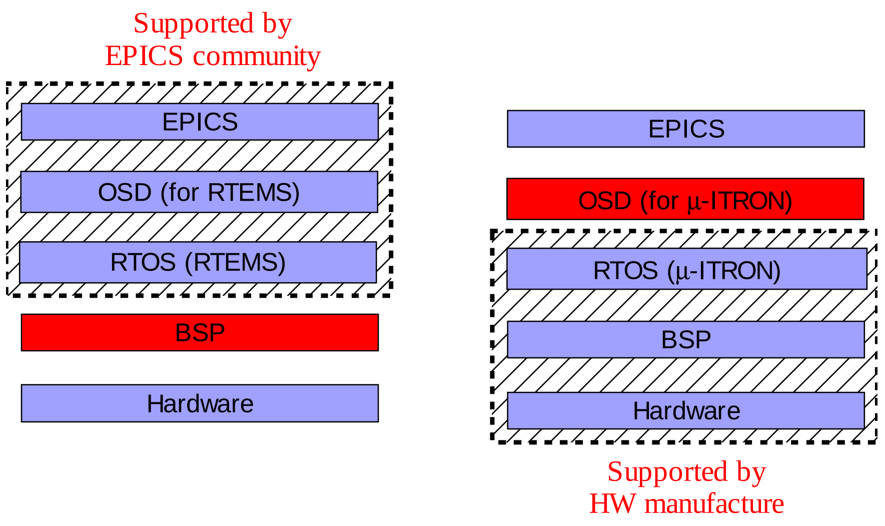

Fig 2. Porting iocCore for µITRON can benefit from utilizing existent BSPs

IMPLEMENTATION

This section describes the target hardware/software and the development environment used for the porting. Some fixes of technical problems found in the porting are also described.

Target hardware/software

As shown in Fig3, we chose Multi Control Unit (MCU) made by Nichizo Electronic & Control Corporation (NDS) as our hardware platform.

The building blocks of the software required to run iocCore on the MCU are shown as follows:

- Kernel – we chose NORTi since three devices listed in the table 1, including MCU, are using it.

- TCP/IP protocol – we chose KASAGO TCP/IP protocol stack from Elmic Systems, Inc..
- BSP – NDS supplies the BSP with the hardware.

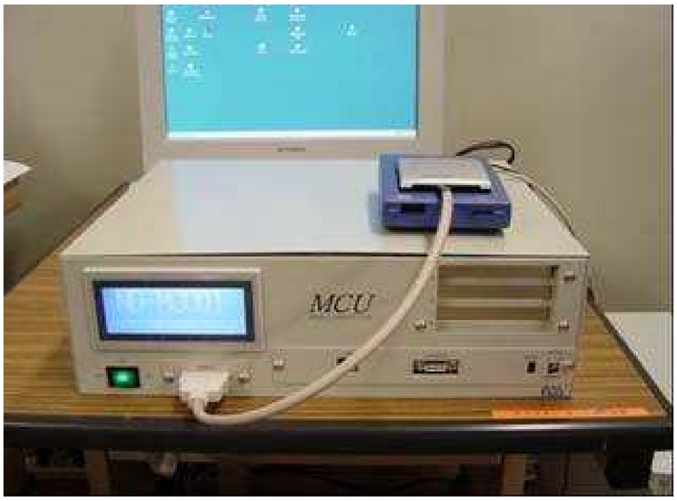

Fig 3. MCU by NDS

Development environment

We chose the following products for the build tool chain and the Inline Circuit Emulator (ICE) for debugging so that the environment matches with that of used to develop the BSP of MCU.

- Super Hitachi Compiler (SHC, Version 8.0.0) with Hitachi Embedded Workshop (HEW)
- PARTNER-Jet: A J-TAG debugger

Building problems of C++ code

Recent versions of EPICS base include lots of C++ code, in which a feature named as Run-time Type Identification (RTTI) was mainly used to identify the object type in exception handler. A switch of SHC on RTTI needed to be turned off since objects compiled with RTTI being turned on are not allowed to be registered to a library.

Though SHC supports Exception basically, there are some missing sub-classes to support Exception. We have implemented those sub-classes required to compile EPICS.

Implementation of OSD libraries

We have implemented OSD libraries, which are composed of several files, for example, osdEvent.c, osdMutex.c, osdThread.c, osdTime.cpp, etc. The NORTi native APIs allowed us to implement OSD libraries just by making wrapper functions around the APIs.

Standard Input/Standard Output on TCP/IP

The iocCore program has its own shell, iocsh, which allows users to interact with the program. The target device, MCU, however, do not have standard input/output as many of embedded controllers. We hence designed and implemented a standard input/output on TCP/IP connections on the Ethernet port of the target. As a result, users can use commands provided with the iocsh program to investigate the status of database records loaded on the IOC and various threads running in the program. It also makes it considerably easier to develop programs on the IOC, such as device/driver support modules and so on.

In addition, we have created another TCP/IP channel in order to download database files from a remote host to the IOC by using a TFTP-like simple protocol. Together with the standard input/output, we can boot up the IOC by using a start-up script file on the remote host as we do with usual VME-based IOCs.

Each of the three channels, standard input, standard output and the file transfer channel, has its own task that handles incoming or outgoing streams of data. A thread on the remote host is supposed to communicate with one of those three tasks on the corresponding channel.

REAL-TIME PERFORMANCE

In a conventional Ethernet-based system, shown in the left side of Fig 1, all of the control logic needs to be implemented on the Ethernet-based controllers if real-time responsiveness is needed. Otherwise, the control logic is separated on both sides of the IOC and the Ethernet-based controller communicating each other over the network, which cannot ensure real-time responsiveness in any case.

On the contrary, provided with a real-time OS, such as µITRON, Embedded EPICS can ensure real-time responsiveness since all of the control logic can reside on a single controller. We have measured the real-time responsiveness of µITRON-based target to confirm that it has this desirable feature.

Method of Measurement

The outline of jitter test is shown in Fig 4:

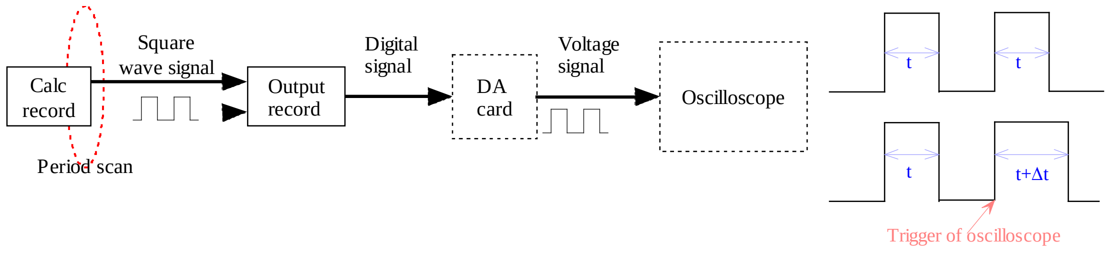

Fig 4: Outline of jitter test

In Fig4, included in the dashed pane are on the target, MCU. A simple database comprised of an Analogue Output (AO) record and a Calculation (Calc) record was created for this test. The Calc record was being processed by periodic scan to produce a square wave signal. The AO record, being processed by Calc record through a forward link, got the raw value from the Calc record to drive the D/A card. The output signal of the D/A card was monitored by using an oscilloscope. The signal, which was shown in the right side of Fig 4, changed its width jittering. The jitter arose from the unpredictable latency of the system due to various causes, such as protection of critical sections from racing tasks, and limited resolution of the timer that wake up the task sleeping for the periodic scan. We measured the jitter with changing the value of SCAN field of Calc record from 0.004 second to 0.5 second.

Table 2 shows the hardware/software specifications of the target, MCU, with a D/A card installed. We have repeated the same measurement on a PC running Linux for a comparison. The specifications of the PC are also shown in Table 2.

	µITRON platform	**Linux platform**
CPU	200MHz SH4	3.2GHz Pentium4
DRAM	16M	1G
DISK	64M Flash ROM	40G Hard disk
DA card	PCI-360116	DA12-4
Tick interval	1 millisecond	1 millisecond
Kernel version	NORTi4	Kernel 2.4.22

Table 2: Specifications of the platforms

For both of μITRON and Linux, we have configured the kernels with the tick interval of 1 millisecond for a better resolution of the timer.

Analysis of jitter test results

Fig 5 shows the measured jitters on both μITRON and Linux in a preliminary test. The horizontal axis is the logarithm value of scan period and the vertical axis is the time offset, which is the deviation from the scheduled time. The length of the vertical line at each measured point shows the range of the jitter.

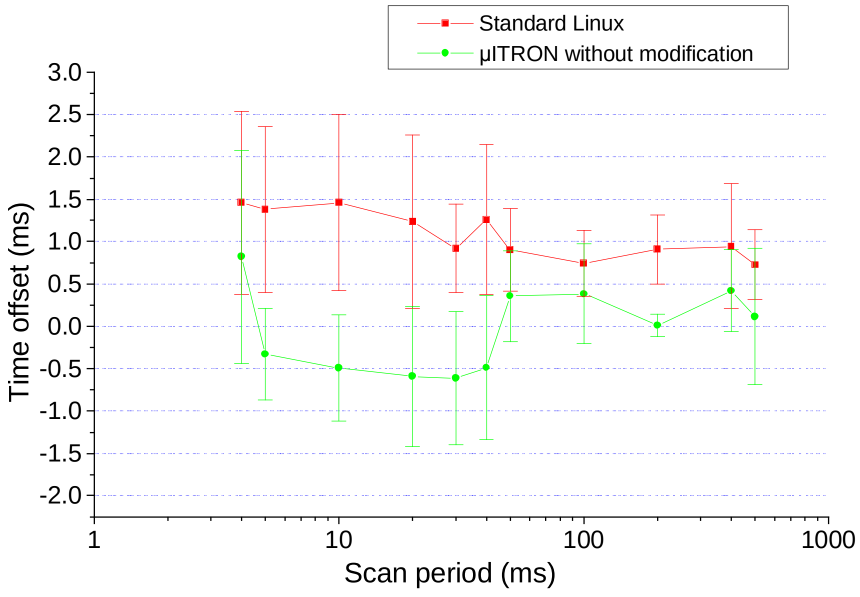

Fig 5: Comparison of jitter between μITRON and Linux
without modification of EPICS code

As shown in Fig5, we found the jitter on μITRON, which should have hard real-time responsiveness, was a few milliseconds against our expectation of within one tick that arises from the timer resolution. We hence investigated the EPICS source for the cause to find a problematic code in a function that calculates the difference between two specified times.

```
epicsTimeDiffInSeconds (const epicsTimeStamp *pLeft, const epicsTimeStamp *pRight)
{
        ……
        return epicsTime (*pLeft) - epicsTime (*pRight);
        ……
}
```

In this function, epicsTime is a class defined in EPICS. The function, which creates and destroys the instances of epicsTime, causes considerable amount of overhead in calculating the time difference itself.

Since the purpose of the time difference calculation was to specify the duration for the scan task to sleep, the overhead of the calculation itself brought in an error of one tick or so. We fixed this problem by changing the function to calculate the time difference so that it does not create and destroy the objects and repeated the same measurement. Fig 6 shows the result with the modified function. With that modification, the jitter was found to be, in most cases, within the range of the resolution of the timer to meet our expectation of the μITRON kernel being a hard real-time one.

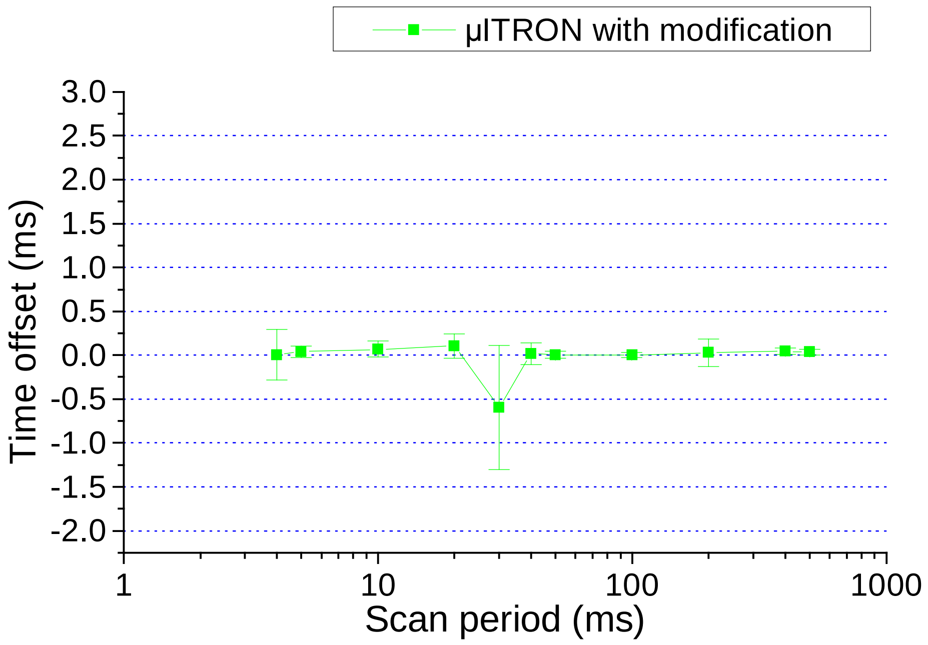

Fig 6: Jitter of µITRON with modification of EPICS code

CONCLUSIONS

We have ported EPICS iocCore onto a µITRON/SH4-based controller in order to investigate a possibility of achieving Embedded EPICS on various types of commercial controllers. We implemented EPICS OSD libraries for µITRON, compiled EPICS base by using the Super Hitachi Compiler, and got the iocCore running on the target successfully. This result proved the feasibility of Embedded EPICS on µITRON/SH4-based controllers.

By using this fully functional IOC, we measured the jitter of periodic execution of a task in order to evaluate real-time performance of Embedded EPICS on µITRON. We found a problem that is related with time calculation in EPICS and fixed it. With that modification, the measurement of the jitter showed that Embedded EPICS on µITRON has the real-time responsiveness as expected.

REFERENCES

[1] G. Jiang, J. Odagiri, N. Yamamoto, et al., "Porting EPICS Core Program onto micro-ITRON/SH4-based Device Controllers", PCaPAC 2005, Hayama, Japan, Mar. 2005.
[2] J. Chiba et al., "A Control System of the Joint-Project Accelerator Complex", ICALEPCS'2003, Gyeongju, Korea, Oct. 2003.
[3] M. Komiyama et al., "Control System for the RIKEN Accelerator Research Facility and RI-Beam Factory", the 17th International Conference on Cyclotrons and Their Applications, Tokyo, Oct. 18-22, 2004.
[4] K. Furukawa, et al., "Network based EPICS Drivers for PLCs and Measurement Stations", ICALEPCS'99, Trieste, Italy, 1999, p409.
[5] M. Kraimer et.al, "EPICS: Porting iocCore to Multiple Operating Systems," ICALEPCS'99, Trieste,Italy, Oct. 1999.
[6] G. Waters, et.al, "TRIUMF/ISAC EPICS IOCs Using a PC104 Platform", ICALEPCS'2003, Gyeongju, Korea, Oct. 2003.
[7] W. Eric Norum, "How to create a simple ColdFire and Altera FPGA IOC (Draft)", http://www.aps.anl.gov/epics/base/RTEMS/FPGA_IOC.pdf, August 24, 2005.
[8] http://tron.um.u-tokyo.ac.jp/TRON/ITRON/home-e.html.