# PORTING EPICS CORE PROGRAM ONTO MICRO-ITRON/SH4-BASED DEVICE CONTROLLERS

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

*Abstract*

Experimental Physics and Industrial Control System (EPICS) is widely used for many accelerator control systems. The most important component of EPICS is iocCore, which is the core software running on Input/Output Controllers (IOCs). In modern accelerator control systems, more and more intelligent controllers with an Ethernet interface, such as PLCs and custom device controllers, at the interface layer below VME single-board computers or PCs working as IOCs. However, in most cases, iocCore can run directly on the intelligent controllers themselves since they run a real-time kernel on a high performance CPU and tens of mega-bytes of memory. Running iocCore directly on the device controllers can reduce the depth of hierarchy in the system to make it simpler and more robust. As a first step towards this scheme, we have ported iocCore onto a target running a μITRON real-time kernel on an SH4 CPU. The technical details of the porting are described.

## 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 to replace those field-busses with Ethernet. Actually, new accelerator projects such as J-PARC [1] and RIBF [2] are going to adopt intelligent device controllers listed in Table 1.

In table 1, MCU and e-RT3 are commercial products available on the market. EMB-LAN100 [3] and N-DIM [4] are custom controllers. EMB-LAN100 was developed by KEK for the control of power supplies of DTL Q-magnets. N-DIM was developed by RIKEN for general purpose control and monitoring.

Using Ethernet as a kind of field-bus has the following benefits:

- Continuity – we can expect continuity in the future because TCP/IP is a widely used standard in various fields
- Well-established – TCP/IP is a well-established technology
- Well-known – Ethernet and TCP/IP are popular and the knowledge of their protocols is widespread
- Flexible – Ethernet can be extended easily and devices can also be added onto it easily

Table 1: Characteristics of some typical network-based controllers

| Controller | Kernel | CPU | RAM (min) |
|---|---|---|---|
| MCU | μITRON | SH4 | 64M |
| e-RT3 | μITRON | SH4 | 32M |
| EMB-LAN100 | μITRON | SH3 | 8M |
| N-DIM | μITRON | SH4 | 6M |

## EMBEDDED EPICS

The left side of figure 1 shows the EPICS-based control system. The Operator Interface (OPI) tools communicate with IOCs through Channel Access (CA) protocol over the network. If only network-based controllers are used and connected to the network in this system, expensive VME computers end up being used just to convert proprietary protocol of each device to EPICS CA protocol, leaving all VME slots unused. We can avoid this situation by running iocCore on each network-based device controllers. In fact, there have been examples of running EPICS iocCore directly on processor cards based upon the PC104 standard, which are running vxWorks or Linux [8, 9].

This scheme, hereafter referred as Embedded EPICS,

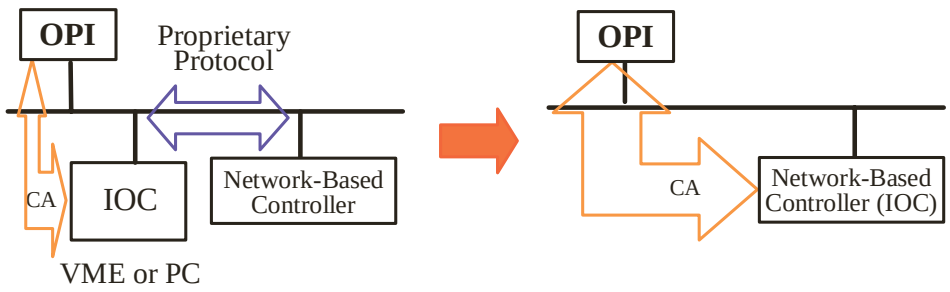


Figure 1: Structure of exclusive EPICS controller and embedded EPICS controller.

has following advantages:

- More distributed –run-time database records are distributed over the network-based controllers rather than concentrated on a single IOC
- More robust – since each network-based controller works as an independent IOC, failure of an IOC localizes to have smaller impact on the system
- Synchronous driver – complicated asynchronous drivers are not necessary

In order to achieve the scheme, the device controllers need to have enough CPU power and memory to run iocCore.

A decade ago, iocCore used to be running on Motorola 68k based CPUs with a few tens of mega-hertz of clock frequency. It is clear that CPU power of the controllers listed in table 1 is enough because an SH4 CPU operates at hundreds of mega-hertz of clock frequency.

As to capacity of RAM, we need a more detailed consideration since some of the device controllers have lower capacity than that of old VME CPU boards. Typically, binary file size of iocCore (EPICS 3.13) is around 1 mega-byte or less. The typical size of vxWorks kernel is also about the same size. The size of record, device and driver support should be small since the only ones that are related with the device controller itself are included. The number of database records is also small for that reason. Several mega-bytes of memory will be sufficient to run IOC core program even if we take additional memory required for communication with CA clients into account. All of the device controllers listed in Table 1 meet the requirement.

As shown in the Table 1, μITRON is used for all of them. μITRON is a real-time, multitasking OS specification intended for use in industrial embedded systems, which is registered with the TRON (The Real-time Operating system Nucleus) Association in Tokyo University in 1984. It is assuming a position as the world's first standard kernel specification [5].

## IOCCORE ON MICRO-ITRON

In addition to hardware capability, the functional capability of software platform is another issue to be taken into consideration.

### *Basic kernel services*

The real-time kernel under iocCore must support multitasking including synchronization, communication and mutual exclusion between tasks and so on. Whether if it has all the functions required to implement Operating System Dependent (OSD) libraries [6], which interface iocCore with the kernel, is to be considered. By investigating the specification, we found that μITRON kernel meets this requirement.

In addition, μITRON has real-time feature, which ensures predictable responsiveness to external events for our purpose [7].

### *TCP/IP support*

Though μITRON has its own specification of Application Interface (API) of TCP/IP protocol stack, EPICS requires a BSD compatible socket interface for network communication. If we use free software for μITRON kernel and others, we have to port a free implementation from elsewhere. It is somewhat a costly work. On the other hand, if we choose to rely on the commercial software, there are several commercial products of socket libraries available on the market.

### *Board Support Package (BSP)*

Once iocCore is ported onto a μITRON-based target, it basically can run on other targets running μITRON. On the other hand we have to develop a BSP every time when we support a new target. Development of BSPs is also a costly and time-consuming task. If we choose free software, we have to develop BSPs for each of the target devices on our own. We have to avoid the cost in order to make Embedded EPICS realistic. To use BSPs as they are from the suppliers, we decided to rely on one commercial product for this reason.

## IMPLEMENTATION

Selecting the proper development environments of software and hardware is crucial to make porting straightforward.

### *Target hardware*

We chose Micro Control Unit (MCU) made by Nichizou Electronic Control Corporation (NDS) as hardware platform, since JAERI had successfully ported CA protocol on μITRON TCP/IP API, with taking CA-part out of iocCore on MCU.


Figure 2: Micro Control Unit (MCU) made by Nichizou Electronic Control Corporation (NDS) .

### *Target Software*

Table 2 shows the building blocks of the software required to run iocCore on the MCU.

Table 2: Target software

| Component | Product | Supplier |
|---|---|---|
| Kernel | NORTi 4.0 | MISPO |
| TCP/IP | KASAGO | Elmic Systems, Inc. |
| BSP | | NDS |

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., and asked them to port it onto MCU. Two libraries, one is for interfacing their TCP/IP protocol stack with μITRON/LAN controller driver, and the other is for interfacing their TCP/IP protocol stack with NORTi

### *Development environment*

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

- Super Hitach Compiler (SHC, Ver. 8.0.0) with Hitachi Embedded Workshop (HEW)
- PARTNER-Jet

Recent versions of EPICS base include lots of C++ codes, in which a feature named as Run-time Type Information (RTTI) was mainly used in exception handlers. The switch about RTTI in SHC needed to be turned off to workaround a problem of unresolved external symbols upon linkage.

SHC is a cross-compiler for embedded system running on Windows. Though it comes with an abundant C/C++ library, there are some missing functions required to compile EPICS iocCore.

### *Implementation of OSD libraries*

We have implemented following files of OSD libraries:

- osdThread.c.
- osdMutex.c
- osdEvent.c
- osdMessageQueue.c
- osdInterrupt.c
- etc.

The NORTi native APIs allowed us to implement OSD libraries just by making wrapper functions around the APIs, as showed in follows:

```
epicsMutexLockStatus epicsMutexOsdLock(struct epicsMutexOSD * id)
  {
        ER ercd;
        ercd =loc_mtx ((ID) id);
        if (ercd != E_OK)
        {
              errlogPrintf ("can't lock Mutex \n");
              return epicsMutexLockError;
        }
        return epicsMutexLockOK;
  }
```

### *Testing BSD socket library*

The most essential functionality of iocCore is maintaining communication channels with clients concurrently. For that reason, we have tested KASAGO TCP/IP product with a simple TCP concurrent server where a server task was waiting for connection requests from clients. When the server task received a request, it created a child task that communicated with the client. No problems were found so far. Overload test must be done to confirm the stability. Keep-alive function should also be tested to confirm that a socket is closed on the server side when the connection with the client is lost.

## CONCLUSIONS

We have almost ported EPICS iocCore onto a μITRON/SH4-based device controller in order to investigate a possibility of Embedded EPICS. We implemented OSD libraries, compiled all of the source codes of iocCore successfully, and tested a commercial socket library. The results showed that Embedded EPICS is feasible. In order to confirm the feasibility, we need to get iocCore running on the target and do more long-term tests.

## REFERENCES


[1] J. Chiba et al., "A Control System of the Joint-Project Accelerator Complex", ICALEPCS'2003, Gyeongju, Korea, Oct. 2003.

[2] Y.Yano, et al., "RI Beam Factory Project at RIKEN," Proc. of 16th Int. Conf. on Cyclotrons and their Applications, East Lansing, U.S.A. (2001) p.161.

[3] K. Furukawa, et al., "Network based EPICS Drivers for PLCs and Measurement Stations", ICALEPCS'99, Trieste, Italy, 1999, p409.

[4] M.Komiyama, et al., "Current Status of the Control System for the RIKEN Accelerator Research Facility", ICALEPCS'2003, Gyeongju, Korea, Oct. 2003.

[5] http://tron.um.u-tokyo.ac.jp/TRON/ITRON/home-e.html.

[6] M. Kraimer et.al, "EPICS: Porting iocCore to Multiple Operating Systems," ICALEPCS'99, Trieste,Italy, Oct. 1999.

[7] Real-time Multitasking OS based on μITRON 4.0 NORTi4 User's Guide.

[8] G. Waters, et.al, "TRIUMF/ISAC EPICS IOCs Using a PC104 Platform", ICALEPCS'2003, Gyeongju, Korea, Oct. 2003.

[9] 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.