THE PXI AND VMEBUS SUPPORTS FOR THE LINUX BASED EPICS

K. H. Kim, M. K. Kim, M. K. Park, J. W. Choi, I. S. Choi, M. C. Kyum, M. Kwon
and the KSTAR Control Team

Basic Science Institute, 52 Yeoeun-Dong, Yusung-Gu, Daejeon 305-806, Republic of Korea

ABSTRACT

Conventionally, Experimental Physics and Industrial Control System (EPICS) has been used on the VxWorks Real-Time Operating System (RTOS) before EPICS release R3.14 which was developed with Operating System Independent (OSI) technology. Now, it can support other Operating Systems (OSs). However, EPICS can operate with other OSs just being used as a softIOC, i.e. without hardware modules, because even now it does not have real hardware support on other OSs. Recently, some of the Board Support Packages (BSPs) for Real-Time Executive for Multiprocessor Systems (RTEMS) Real Time Operating Systems (RTOS) have been made in the EPICS community but they are restricted only to the PowerPC based VMEbus system. Alternatively, we have considered the Linux operating system as a next EPICS operating environment and made some progress on the PCI Extensions for Instruments (PXI) and VMEbus supports. We made genericPXI device supports which can support almost all of National Instrument's Data Acquisition (DAQ) modules based on PCI, compact PCI (cPCI), and PXI. Additionally, we also have a plan to extend it to support Timing and Signal conditioning modules. We also made the vmeUniverse driver support which is a VMEbus support. It provides application programming interface (APIs) functions similar to VxWorks and the configuration features based on IOC shell commands. We have provided it to the KEKB and JPARC control groups in Japan and they examined it to measure the performance of interrupt handling. The results satisfied their requirements to adopt it as a part of their control system. We will describe the details of implementations of both genericPXI as well as vmeUniverse and will present the test results of vmeUniverse.

INTRODUCTION

We are considering using a PXI based data acquisition system for some of the KSTAR [1, 2] machine diagnostic systems because the PXI system provides various I/O modules and commercial adaptive signal conditioning modules. These modules have fast bus speed and show enough performance to handle a large number of input/output (I/O) points. However, until recently, the PXI system has been supported only by LabView software provided by National Instruments (NI). In order to use EPICS instead of LabView, we have to develop a software layer to interface between EPICS and PXI hardware. Hence we started to develop driver supports and device supports named genericPXI. The genericPXI contains the drvgenericPXI driver support layer and the devgenericPXI device supports.

We are also considering the use of the Linux based VMEbus system for some of the KSTAR machine control systems. Those systems do not require a real-time operating system such as the vxWorks but need to use VMEbus modules. We have developed driver support software, namely vmeUniverse, which provides the VMEbus access features on the Linux platform.

GENERIC PXI SUPPORTS

The *drvgenericPXI* (driver support) is a software layer located in the bottom of EPICS that has access to the PXI hardware through the DAQmxBase/DAQmx libraries. These libraries consist of a set of API's for direct access to the PXI hardware.

This driver can communicate with many kinds of device supports, such as analog input (ai), analog output (ao), binary input (bi), binary output (bo), multibits input (mbbi), multibits output (mbbo), waveforms, etc. – which are conventional EPICS record types. It has command line configuration and status monitoring features. Users can configure PXI hardware by issuing commands on the IOC shell and also can monitor status of the driver by command line functions. It is a useful configuration method in EPICS, because, in most of the cases, these commands are called by start-up script when the

IOC is booted automatically. The driver has two kinds of operation modes: one is the External driven mode and the other is the Continuous acquisition mode.

External driven mode

The External driven mode corresponds to the ao, bo and mbbo types of device support. The sequence of operations is as follows: Events are generated by user threads (e.g., other record processes, Channel Access (CA) events, periodic scanner, and dbAccess) and this activates record support, see Fig. 1. The record support forwards the request to device support, which in turn puts the request into the queue. This processing chain (previous paragraph) runs on the user thread. After putting the request in the queue, the record processing is temporarily on hold, to be finished later. The user thread is now free to run other things.

When the driver thread detects a non-empty queue, it will start hardware processing and will be blocked until the processing is finished. When hardware finishes processing the request, the driver supports will invoke the callback function in the device support. The callback function finishes the record processing. If the driver thread has a heavy load, the callback function in the device support could request one of three callback threads (prioritized callback/general purpose callback tasks) in EPICS to complete the record processing. This decision should be made by the programmer for the device layer when the callback function is built. If the record processing is completed by the driver thread, the structure of the program becomes simpler. However, this sacrifices the response of the driver thread because the driver thread can not respond immediately to the next external events. We can expect faster response if we choose prioritized callback to complete record processing; however, there is a small overhead associated with changing running threads to complete record processing, callback queuing, monitoring queue, context switching, etc. Overall faster response still does compensate for small overheads.

As the previous discussion, the record processing is not completed by just one thread at a given time, there are more than two threads involved, and also more than two execution time slots are involved thus we can call asynchronous processing for the record processing.

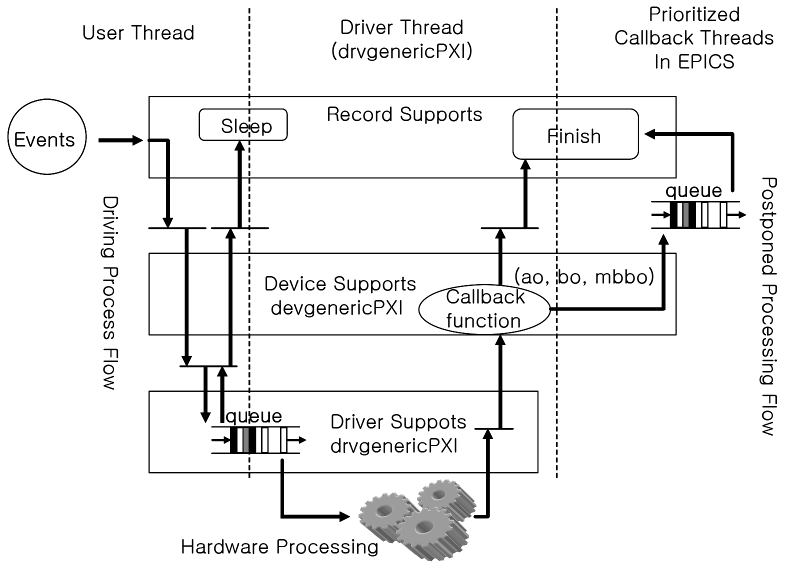

User Thread
Driver Thread (drvgenericPXI)
Prioritized Callback Threads In EPICS
Events
Sleep
Record Supports
Finish
queue
Driving Process Flow
Postponed Processing Flow
Device Supports devgenericPXI
(ao, bo, mbbo)
Callback function
queue
Driver Suppots drvgenericPXI
Hardware Processing

Figure 1: External driven mode

Continuous acquisition mode

The Continuous acquisition mode corresponds to ai, bi, mbbi, and waveform types of device support. There are two types for the continuous acquisition mechanism: driver thread running on an infinite loop (acquisition loop) at a prescribed rate, and a swing buffer mechanism on the Direct Memory Access (DMA) buffer, see Fig. 2. For both mechanisms, the execution rate depends on sampling rate, triggering, and number of data samples.

In the first type, whole processing chains (from driver layer processing to record layer one) are driven by the acquisition loop in the driver layer. This loop has two linked lists (pre-callback and post-callback), each linked list executes callback in the device layer. The pre-callback linked list stores a list of callback functions to execute before starting the hardware acquisition. The post-callback linked list stores a list of callback functions to execute after the acquisition ends. If there is nothing to do before acquisition, the pre-callback linked list is empty. However, if there is some pre-processing to be done, the device programmer can use the pre-callback feature to implement any preprocessing function needed in the device support. The post-callback executes post-processing actions in the device support, and the device support invokes record processing. If the post-processing makes record processing directly, the entire record processing is running on the driver thread. It is not advisable to have a fast execution rate for the acquisition loop. While the driver thread executes record processing, the DMA buffer can overflow if the hardware has a fast acquisition rate. To prevent overflow, prioritized callback for post processing can be used. The post-processing inserts a function pointer to invoke record processing into a prioritized callback queue, so that the driver thread can run again on the acquisition loop. Using prioritized callback results in better performance in the acquisition loops. However, there can be an unpredictable short delay in record processing because the prioritized callback may be processing something else.

In the second type, the record processing could be triggered by events originating from the user thread. In this case, the record processing is synchronous. As shown by the dashed arrow in Fig. 2, record support immediately reads out the acquired data from a buffer in the device layer and completes the record processing. The acquired data could come from the previous step in the acquisition loop.

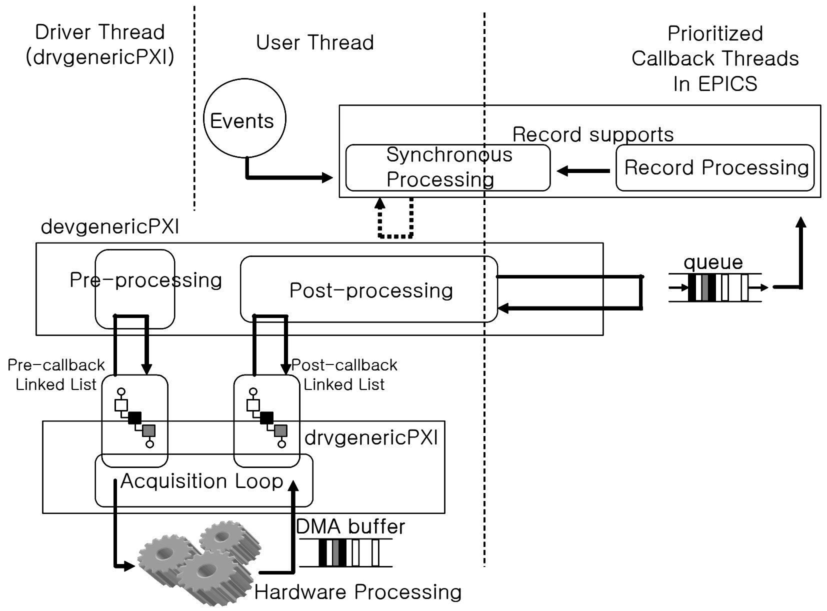
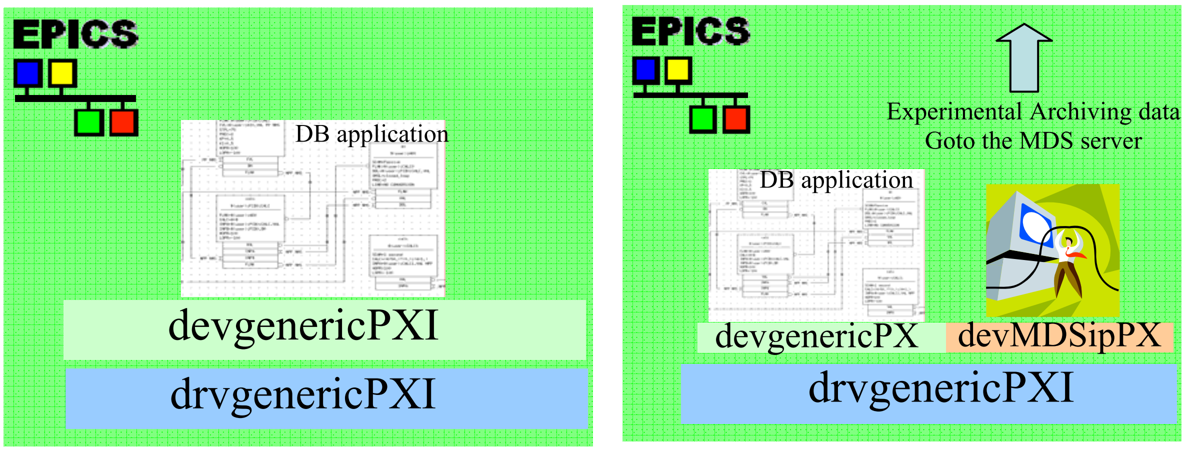


Figure 2: Continuous acquisition mode

Multiple device layer considerations

The devgenericPXI is designed for conventional EPICS record supports which have been developed to correspond to generic machine control and monitoring. However, the genericPXI needs additional capabilities (e.g., streaming and archiving features) because it will be used for continuous data acquisition systems for many of the KSTAR diagnostics systems. It has to handle heavy data transfer to the archiving system without data loss during the acquisition. To achieve this requirement, the drvgenericPXI can work with a special device support (i.e., streaming device support) or can also work with both of them (special device supports and devgenericPXI). We have included some features in drvgenericPXI to handle multiple device supports. One of the features is pre- and post-callback linked lists. There are no limitations to register callback functions which are located in individual device supports. The only limitation is the execution time of the linked lists. We have to reduce the callback execution time as in the previous discussion. The other feature is re-entrancy and thread safety for APIs in drvgenericPXI. Individual device supports which are related with drvgenericPXI can make connection with both of the features.



single device configuration multiple device configuration

Figure 3: software stacks of genericPXI

Portability of the genericPXI

The devgenericPXI doesn't have any knowledge about the PXI hardware and its library, but drvgenericPXI does. However, the driver does not have direct control of PXI hardware. Direct access to PXI hardware is done through the DAQmxBase library, provided by NI. Actually this will be replaced by DAQmx. NI started support for the DAQmxBase in the Linux environment, but this seems to be a small pilot project, supporting only two kinds of NI hardware, the E series and M series. Therefore, NI wants to move to DAQmx because this software is their base software chassis, supports almost all NI hardware and is platform independent. If they move to DAQmx, little or no change needs to be made to drvgenericPXI. Since the DAQmxBase library has almost the same structure as DAQmx, genericPXI can support most NI hardware.

COMPACT FIELD POINT SUPPORTS

We now discuss other aspects of the control requirements. Many of control systems, such as the vacuum and cryostat controllers, need robust and cheap I/O modules that are not necessarily fast. There is no need for either fast response or heavy computational capability similar to the VME system. The requirements specify the use of remote I/O modules. The EPICS softIOC and NI's CompactFieldPoint (cFP) solution satisfied the design requirements. The cFP has a small CPU module for the RT-LabView program on PharLab RTOS and also has a variety of I/O modules – ai, ao, di, and do. We have developed a small protocol based on Transmission Control Protocol/Internet Protocol (TCP/IP) to enable communication between EPICS and cFP. One half of the protocol was developed on RT-LabView to execute on a cFP CPU module. The other half of the protocol was developed in EPICS, namely, drvcFP20x0 and devcFP20x0. This protocol has a self configurable feature. When the communication is established, the cFP notifies EPICS about their hardware configuration (e.g., module types, order of modules, number of channels for each module, etc.), then EPICS decides how to parse the message from cFP and how to address individual channels in the cFP I/O modules. We have implemented cFP and the above protocol with Linux based softIOC for our vacuum system prototype. We can control and monitor all of our vacuum valves and pumps with this protocol.

We hope that the cFP remote I/O solution with the communication protocol can replace some of Programmable Logic Controller (PLC) hardware. We now can use soft IOC with State Notation Language (SNL) programs instead of PLC CPU modules and ladder programs. The SNL program is a native feature of EPICS, it has a powerful state programming language, deep coupling with EPICS, simplified development, easier maintenance, and relieves the need for a PLC engineer. Moreover, the cFP remote I/O is cheaper than PLC modules.

VME UNIVERSE SUPPORTS

We have made vmeUniverse driver supports which provide the possibility to utilize VMEbus on the Linux platform. The vmeUniverse is composed of two software components which are the drvvmeUniverse EPICS driver support and the vme_universe Linux kernel module. The drvvmeUniverse is located on the bottom of EPICS IOC and provides the user level APIs and various IOC shell commands. The vme_universe Linux kernel module performs more lower level system services such as the VME interrupt handling, VMEbus address mapping into the virtual address space, DMA control features, and other low level functions related to the PCI-VME bridge chip set.

The vmeUniverse provides a set of APIs which are similar to vxWorks' APIs related to VMEbus access. Thus, these APIs can help in reducing the porting efforts if we try to port the existing EPICS driver/device support from vxWorks to the Linux platform. As an example, we just changed several lines in the source code of VMIC3122 driver/device supports which is existing software on vxWorks to port them to the vmeUniverse environment under the Linux. The vmeUniverse provides not only the cross platform portability as mentioned above but also a flexible configuration method. In the vxWorks environment, if you want to change the range of VMEbus window, you have to modify some parameter in your board support package (BSP) and then have to re-build your vxWorks image again. However, the vmeUniverse does not require either a re-compiling or a re-building process to apply new changes. The vmeUniverse provides a set of IOC shell commands to configure itself. Thus, you just change a few lines in the start-up script and re-start IOC to apply the new changes. Actually, the vmeUniverse is running on the Linux platform which is not a real-time environment. So, we are interested in the interrupt latency because it should have an un-predictable time delay. Actually, the user defined VMEbus interrupt handler could not be running on the kernel privileged mode, because it is defined in user code and must be running on user mode threads. So, the vme_universe kernel module has a low level interrupt handler to handle VMEbus interrupts directly. Then the low level handler generates POSIX signals to notify interrupt events to the user thread which contains the user defined interrupt handler. There are at least two kinds of delays involved in the total interrupt latency. These are the native interrupt latency by the low level interrupt handler in vme_universe kernel module and the overhead due to POSIX signal handling. In the worst case, one more delay should be involved, that is the overhead of scheduling and context switching. If the user thread was pre-empted by another thread, the user defined handler has to do the scheduling. According to our measurement, the total latency was less than 60 μs. However, if the latency is larger than a few tens of times compared to vxWorks, it is still a reasonable value to apply it to non hard real-time applications.

Now the vmeUniverse is still in the developing phase. We have finished developing the following features.

- Create a Master/Slave VMEbus address window on the virtual address space
- Delete the Master/Slave VMEbus address window on the virtual address space
- Configure a VMEbus interrupt (Enable/Disable)
- Register a VMEbus interrupt handler on the user thread
- Generate a VMEbus interrupt
- Report features to check up on the Tundra PCI-VMEbus bridge chip set
  - Display status register on the UniverseII bridge chip set
  - Display the I/O memory map
  - Display the status of the master/slave window
- Command line VMEbus access: PEEK/POKE/DUMP command on the IOC shell

Furthermore we will complete developing following features soon.

- The DMA mastering features
- Location monitoring interrupts

Figure 4 shows the structure of vmeUniverse driver supports.

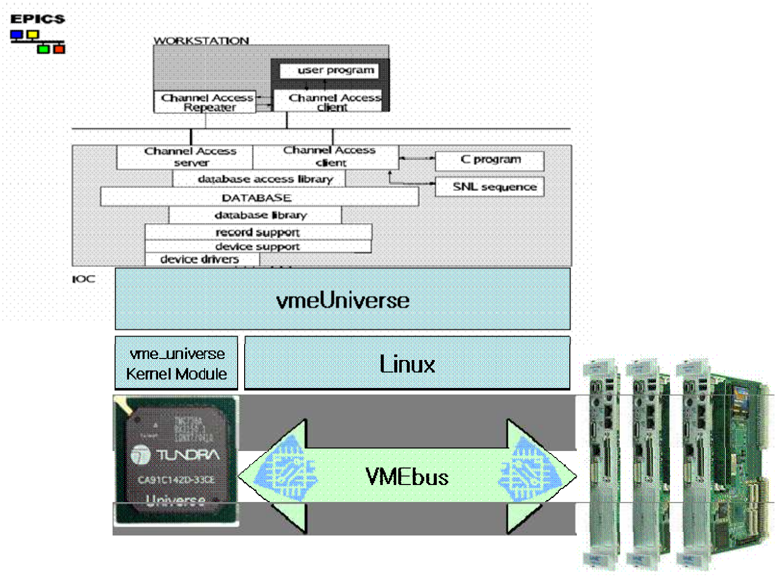


Figure 4: vmeUniverse driver supports

CONCLUSIONS

We have discussed the EPICS PXI support which is device/driver software running on NI's DAQmxBase abstraction layer. It will support almost all NI hardware on the PXI/PCI bus without any modification after NI releases their DAQmx library to replace DAQmxBase. The device/driver now has a boot configuration feature, configured by IOC shell commands and start-up script. However, we plan to upgrade it for run-time configuration in the near future. Then, we can arrange the hardware configuration on the fly through the EPICS records which are related to the run-time configuration feature. We also discussed the NI cFP remote I/O solution with our communication protocol. This was successfully implemented in the vacuum prototype controller for the KSTAR project. We hope it can replace PLC hardware in the KSTAR control system. We can use EPICS soft IOC with the SNL program instead of the PLC CPU module and ladder program. Thus, the development is simplified and maintenance becomes easier. We have also discussed the vmeUniverse support which is device/driver software running on Linux platform. It makes VMEbus access and provides very similar VMEbus access APIs with the vxWorks. Thus, we can easily port existing hardware drivers from vxWorks to the Linux platform. We hope that the vmeUniverse and Linux platform will be used for non real-time applications in the KSTAR project instead of the vxWorks platform. Actually, the vmeUniverse support has been provided to the JPARC and KEKB control group to be used for JPARC project. They already made a performance test for the vmeUniverse and wrote a report [3]. Now we are implementing the DMA features in the vmeUniverse by asking for help from the KEKB control group.

ACKNOWLEDGEMENTS

This works was supported by the Korean Ministry of Science and Technology. Special thanks to Guobao Shen in KEKB for providing us with a modification of the vme_universe kernel module which is related to VMEbus interrupt handling. We also acknowledge substantive technical discussion with A. C. England, KBSI.

REFERENCES

[1] G. S. Lee, J. Kim, S. M. Hwang, et al., "The KSTAR Project: Advanced Steady-state Super-conducting Tokamak Experiments" 17th IAEA Fusion Energy Conference, Yokohama, Japan, Oct. 19-24, 1998, and "The design of the KSTAR Tokamak Engineering" Fusion Engineering and Design, Vol. 46 Issues 2-4 405 (1999).
[2] M. Kwon, J. S. Bak, G. S. Lee, "Progress of the KSTAR Tokamak Engineering," Fusion Science and Technology 42, 167 (2002).
[3] JPARC control group and KEKB control group, Linux-Based IOC by using Intel Based VME-SBC, KEK-preprint 2005-14