Drivers and Software for MicroTCA.4

Martin Killenberg, Sebastian Marsching, Ludwig Petrosyan, Adam Piotrowski, Christian Schmidt

***Abstract*—The MicroTCA.4 crate standard provides a powerful electronic platform for digital and analog signal processing. The crate standard is highly configurable and due to an excellent hardware modularity rapid adaption to various different applications is possible. Besides the hardware modularity, it is the software reliability and flexibility as well as the easy integration into existing software infrastructures that will drive the widespread adoption of the new standard.**

The DESY MicroTCA.4 User Tool Kit (MTCA4U) provides drivers, and a C++ API for accessing the MicroTCA.4 devices and interfacing to the control system. The main focus of the tool kit is flexibility to enable fast development. It uses a universal, expandable PCIexpress driver for all devices developed at DESY. An interface layer with callback functions allows to decouple the application code from the control system, making the software easy to adapt for the use at different facilities. In addition, applications like the Low Level Radio Frequency (LLRF) control at the European XFEL and FLASH require guaranteed response times of the system. The driver and the high-level software are currently being reviewed to fulfill these requirements.

We present the design of the MTCA4U Tool Kit and report on the activities to add real-time capability.

I. Introduction

THE MicroTCA.4 crate standard [1][2] provides a platform for digital and analog data processing in one crate. It is geared towards data acquisition and control applications, providing a backplane with high-speed point to point serial links, a common high-speed data bus (PCIexpress in this case) as well as clock and trigger lines. In typical control applications large amounts of data have to be digitized and processed in real-time on the front end CPU of the MicroTCA.4 crate. The trigger signal being distributed via the backplane of the crate allows for the user process to synchronize directly with the readout hardware. This minimizes latencies as no external synchronization is required.

A. MTCA4U—The DESY MicroTCA.4 User Tool Kit

The main goal of the DESY MicroTCA.4 User Tool Kit (MTCA4U) is to provide a library which allows efficient, yet easy to use access to the MicroTCA.4 hardware in C++. The design layout of the tool kit is depicted in figure 1.

Manuscript received May 22, 2014. This work is supported by the Helmholtz Validation Fund HVF-0016 MTCA.4 for Industry.

M. Killenberg, L. Petrosyan and C. Schmidt are with Deutsches Elektronen-Synchrotron DESY, 22607 Hamburg, Germany (e-mail: martin.killenberg@desy.de, ludwig.petrosyan@desy.de, christian.schmidt@desy.de).

S. Marsching is with aquenos GmbH, 76532 Baden-Baden, Germany (e-mail: sebastian.marsching@aquenos.com)

A. Piotrowski is with FastLogic Sp. z o.o., 90-441 Łódź. Poland (e-mail: apiotrowski@fastlogic.pl).

II. The Linux Kernel Module

The Linux kernel module (driver) provides access to the MicroTCA.4 devices via the PCIexpress bus. As the basic access to the PCIexpress address space is not device dependent, we follow the concept of a universal driver for all MicroTCA.4 boards. The kernel module uses the Linux Device Driver Model which allows module stacking, so that the driver can be split into two layers: A universal part provides all common structures and implements access to the PCIexpress I/O address space. The device specific part implements only firmware-dependent features like Direct Memory Access (DMA), and uses all basic functionality of the universal part. For all devices developed at DESY the firmware will provide a standard register set and the same DMA mechanism, which permits to use a common driver for all boards. For devices from other vendors the universal part enables out-of-the-box access to the basic features, which can be complemented by writing a driver module based on the universal driver part. Like this the interface in MTCA4U does not change and the new device is easy to integrate into existing software.

III. The C++ Device API

The basic high level API provides C++ classes which allow access to all the functionality provided by the kernel module without requiring the user to have knowledge about implementation details like IOCTL sequences. In addition, it will have a callback mechanism for hot-plug events, which allows the application to go to a safe state.

A main component of the C++ library is the register name mapping. With evolving firmware the address of a register can change in the PCIexpress I/O address space. To make the user code robust against these changes, the registers can be accessed by their name instead of using the address directly, which also improves the code readability. The required mapping file is automatically generated by the Board Support Package together with the firmware. Performance overhead due to repeated table look-up is avoided by the use of register accessor objects, which cache the address and provide fast access to the hardware. Currently the mapping file implementation is being changed from plain text files to XML. This allows for more flexibility and enables new features like automatic data type conversion, for instance fixed point to floating point or signed 24 bit integer to system types.

A. Separating the Business Logic from the Control System

For larger control applications the different components use a control system or a middleware layer to communicate with each other. For this purpose the control system provides data structures which are used inside the user code. This causes a strong coupling of the code to the control system.

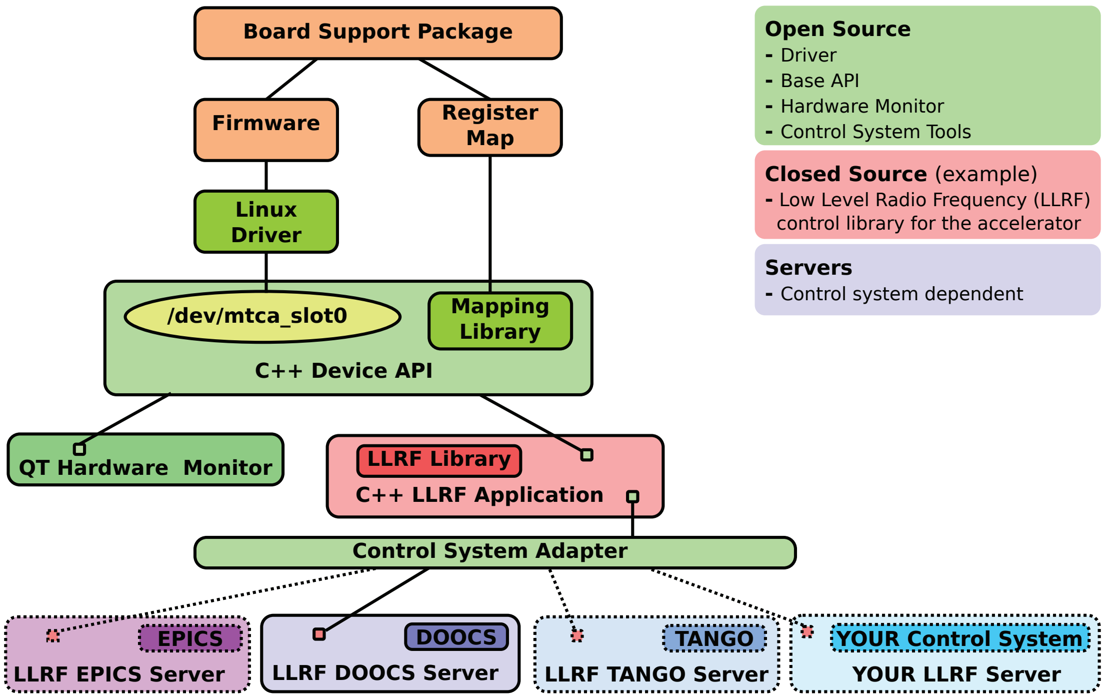

Fig. 1. The design concept of the MicroTCA.4 User Tool Kit MTCA4U.

To make applications portable between control systems, the application code has to be equipped with an adapter layer. This layer should be as thin as possible. Features like variable history and network communication, which are provided by the control system, should not be duplicated in MTCA4U. In addition, the copying of large data structures from the application to the control system should be avoided for performance reasons. MTCA4U will implement process variables which provide access to the content, as well as callback functions which can trigger message sending for message driven control systems. For large data structures these variables will be templated adapters to control system classes, so the application is interfaced to the control system by recompilation without the need to modify the source code. It is important to note that the process variable interface will not provide any control system specific functionality, except to register callback functions which are executed when the variable content is changing. This is important to keep the adapter layer thin and independent from the different control systems. A full adapter to all the features provided by control systems, like history and external synchronization, would be a major effort and require either all control systems to provide the implemented features, or to implement them in the adapter. If the list of features is reduced to common functionality, it would be very limited and probably not provide a useful set of tools at all. In addition adding new control systems would be restricted to those systems which also provide these features. But implementing the missing functionality has to be avoided because it is a large amount of work, does not scale and is basically impossible to maintain.

This is why the approach is to keep the business logic agnostic to the fact that it is running in the context of a control system. It requests process variables from a factory and treats them like normal data types. The concrete factory implementation is specific for each control system. But once this factory implementation has been written, all applications using the MTCA4U control system adapter can be used with this control system without additional adaptation. Whenever the business logic has to react on signals or state changes from the control system, this is implemented using callback functions, which are also registered using the factory. This approach also has an impact on the design of the application code: In principle it has to be able to run without a control system attached. The advantage here is that one can run it for testing with a small, local script or GUI without the need to set up a control system environment. In addition it simplifies unit testing.

IV. Use Case and Real-Time Requirements

A typical application of MicroTCA.4 is the Low Level Radio Frequency (LLRF) control of particle accelerators. The LLRF control of the European XFEL at DESY is used as a case study to understand the real-time requirements.

The European XFEL is a linear accelerator running in pulsed mode with a repetition rate of 10 Hz. [3] The task of the LLRF controller application [4] is to read a large amount of data from seven different boards, calibrate the data, calculate new control tables and load them back onto the FPGA boards. In addition it sends out all the recorded and calculated data via the control system to the data acquisition system (DAQ). The calculation of the control tables is the task which should be finished with real-time precision before the next pulse, while the shipping to the DAQ can still go on while the next pulse is already being recorded. In the current implementation everything is done in one loop, so if the currently not real-time capable loop is late, it will not update the control tables and the data for one pulse will be lost.

There are fixed latencies for the trigger to arrive and a jitter of the pulse arrival time has to be taken into account, which leaves 96 ms out of the nominal 100 ms for the control loop on the CPU to calculate and upload the new values. This seems like a soft real-time requirement, especially considering that the fast feedback and control are implemented in the firmware on the FPGA boards. It is the reason why real-time capability has not been addressed yet. But although the accelerator will still run if the correction for a single pulse is missing, or data for one pulse is not shipped to the DAQ, the performance is not optimal. It is important to deliver the new control table in time for the best possible beam stability. In case of errors it is crucial to have all data recorded, and errors are especially likely if the timing is off, which is exactly the situation where the current scenario will loose data.

The calibration of the data and the calculation of the new control table cause a high CPU load of $\approx$90 %. This means that the overall jitter of the computing time, including the hardware access, has to be kept low.

The LLRF system developed at DESY is planned to work not only at the XFEL, but also at other accelerator facilities which will be running different control systems. The approach to use a control system adapter has been presented in section III-A. We will see in the next section that this also can have implications on the real-time behavior.

V. Identifying Real-Time Issues

A first analysis of the code showed three main issues which have to be addressed to make the LLRF application real-time capable:

- The current version of MTCA4U has not been designed for real-time use.
- The communication with the hardware currently is blocking.
- Not all target control systems are real-time capable.

The first item only concerns the usual changes which have to be done to make code real-time capable, mainly preventing dynamic memory allocation in the real-time loop and not sharing mutex protected variables with non-real-time threads.

To allow parallel access of multiple applications to the hardware, the communication with the hardware is locked by a mutex inside the driver. Like this multiple programs can open the device file and access the MicroTCA.4 board simultaneously. Especially for development this is convenient, but the locking prohibits the use in real-time applications.

Control systems usually implement communication threads for the network I/O, which means that the variable content of their data structures has to be accessed in a thread safe way. If the thread safety is realized using a locking mechanism, these classes are not real-time capable. As MTCA4U is supposed to run with a variety of control systems, this issue has to be addressed in the design of the process variable adapter and the application code.

VI. Solution Ideas

The requirements section showed that the LLRF application will have at least two threads:[1] A real-time thread running the control loop and a non real-time thread for shipping data via the control system. As variable changes from the control system are coming asynchronously to the control loop, there is only one point in time where data between the two threads is synchronized.[2] This is done using a triple-buffer pattern with pointers to all required data being contained in one data structure. The code in both threads has to make sure it does not keep copies of pointers to buffers which have been handed over to the other thread. This is always the case when working with a triple buffer, but needs special attention in this case where a structure with many pointers is exchanged between the threads.

The hardware access lock in the driver can be avoided if the real-time application is getting exclusive hardware access and the device file can only be opened once. However, when searching for a problem it is convenient to get direct access to the hardware registers with a separate tool while the control application is running. To allow this, the driver could have an IOCTL command which blocks the device and does not allow concurrent access. In this case locking a mutex could never fail, and might even be deactivated for performance reasons. To allow access for debugging, the LLRF application could be put into a debug mode where the driver is unlocked. The LLRF application looses its hard real-time capability in this case, which can be tolerated when searching for a problem as normal operation is probably perturbed anyway.

A. *The Process Variable Adapter*

As mentioned in section III-A, the process variable adapter will only have data access features in its interface, no control system features. For simple data types this means accessor functions (getters and setters) as well as the on-change callback function. In addition to these methods, arrays will have iterators and random access operators. It is planned to have an interface similar to the C++ std::array.

In order to avoid unnecessary copying, the MTCA4U process variable will contain an instance of the corresponding control system class which is accessed by reference. This means the control system variable will not be real-time capable if the control system uses a locking implementation to ensure thread safety. For this reason it is necessary to copy the data to a buffer which is being exchanged with the real-time thread. This *copy once* approach has to be implemented such that this one, intentional copy is the only one for large data structures in order not to loose too much performance. Copying once is the price that has to be paid for having a control system independent, real-time capable application.

[1]To be exact the LLRF software will have at least two threads in the application part. The communication level of the control system will also introduce threads for network communication, e.g.

[2]Or one point for input at the beginning of the loop and one point for output at the end of the loop.

B. Not Implemented: Real-Time Capable Control System Data Structures

Especially for large data structures it would be desirable to have a non-copying solution for the process variable adapter. This is only possible if it is supported by the control system. Let's assume the control system is using a ring buffer to implement a history function for a large data structure. The ring buffer can be dynamic in size or it contains n pre-allocated buffers. Either way, it is being accessed by the application thread and the network communication, so it will probably be protected by a locking mechanism. It usually is not possible to get access to the raw buffers to use them directly in the application code.

A possible solution could be an $n + 2$ ring buffer. The ring buffer itself contains pointers to n of the buffers. One additional buffer is used as a *transfer buffer*, the second additional buffer is exclusively used by the real-time thread of the application. This works like the normal triple buffer approach used to transfer data from or to a real-time thread: Both the communication thread, which is holding the ring buffer, and the real-time thread can perform an atomic swap with the transfer buffer. The real-time thread requests a buffer on initialization and swaps it with the transfer buffer whenever it has been filled with new data. When the communication gets a read request, it swaps the transfer buffer with the tail (oldest buffer) and checks whether the buffer it received is newer than the head. If so it moves its head and tail pointers accordingly. If not it swaps the tail and transfer buffer again. Now the tail either contains the previous tail or a buffer which is newer than the head. In the latter case the head and tail pointer are updated.

This solution is efficient because it uses pre-allocated buffers and no copying of the data is needed, and it allows large, complex data structures defined by the control system to be used directly in the user application. The algorithm is safe and convenient because the memory allocation is handled by the control system class and not by the application code. In addition it is thread safe and lock-free, which enables the use in real-time context.

The reason this approach is not pursued at the moment is that this functionality has to be provided by the control system. As MTCA4U is intended to work with many control systems, which do not all provide this feature, we are going for a copying solution. It is currently being studied if the design of the process variable adapter can be done such that an $n+2$ ring buffer can be used when available, and a copying solution can easily be implemented otherwise.

VII. Conclusions

The DESY MicroTCA.4 User Tool Kit MTCA4U is a C++ library which allows convenient access to MicroTCA.4 boards via PCIexpress. A control system adapter allows the development of application code which is independent from the actual control system in use. This makes the business logic portable between control systems with minimal effort. We identified the main issues which have to be addressed to make applications using MTCA4U real-time capable. Solution ideas have been presented, which are currently being implemented and tested.

Acknowledgment

Thanks to Michael Heuer, DESY, for his valuable input on real-time systems and triple buffers.

References

[1] PICMG®, *Micro Telecommunications Computing Architecture, MicroTCA.0 R1.0*, 2006

[2] PICMG®, *MicroTCA® Enhancements for Rear I/O and Precision Timing, MicroTCA.4 R1.0*, 2011/2012

[3] M. Altarelli et al., *XFEL : The European X-Ray Free-Electron Laser : Technical Design Report*, DESY-2006-097, DESY, Hamburg, 2007

[4] C. Schmidt et al., *Real time control of RF fields using a MicroTCA.4 based LLRF system at FLASH*, 19th IEEE Real-Time Conference, Nara, Japan, 2014

[5] *MTCA4U—The DESY MicroTCA.4 User Tool Kit*, Subversion Repository https://svnsrv.desy.de/public/mtca4u