ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP025

BOARD BRING-UP WITH FPGA FRAMEWORK AND ChimeraTK ON Yocto*

J. Georg[1†], A. Barker[1], Ł. Butkowski[1], M. Hierholzer[1], M. Killenberg[1], T. Kozak[1], N. Omidsajedi[1], M. Randall[1], D. Rothe[1], N. Shehzad[1], C. Willner[1], K. Zenker[2]
[1]Deutsches Elektronen-Synchrotron DESY, Notkestr. 85, 22607 Hamburg, Germany
[2]Helmholtz-Zentrum Dresden-Rossendorf, Bautzner Landstr. 400, 01328 Dresden, Germany

Abstract

This paper will showcase our experience in board bring-up using our Field Programmable Gate Arrays (FPGAs) Framework (FWK) and ChimeraTK, our C++ hardware abstraction libraries. The challenges involved in working with different FPGA vendors will be discussed, as well as how the framework and libraries help to abstract vendor-specific details to provide a consistent interface for applications. Our approach to integrating this framework and libraries with Yocto, a popular open-source project for building custom Linux distributions, will be discussed. We will show how we use Yocto's flexibility and extensibility to create a customized Linux image that includes our FPGA drivers and tools, and discuss the benefits of this approach for embedded development. Finally, we will share some of our best practices for board bring-up using our framework and libraries, including tips for debugging and testing. Our experience with FPGA-based board bring-up using ChimeraTK and Yocto should be valuable to anyone interested in developing embedded systems with FPGA technology.

INTRODUCTION

New challenges in accelerator operations are frequently met with the introduction of custom-built hardware that can perform time-critical computing tasks on its own. While the bulk of these tasks are usually implemented in Field Programmable Gate Arrays (FPGA), they are not free from the need of user interaction – for example for run-time modification of algorithm parameters or status monitoring. Finally, the acquired data needs to be exposed to the surrounding control environment.

Problem Description

Often enough, the user-facing part of such software is written from scratch for each new piece of hardware. This can lead to a considerable delay in the availability of the hardware to the introduction of the hardware into the control environment, slowing down feed-back loops between users of the hardware and the developers of the on-hardware algorithms. This can delay the uncovering of problems in the overall design happening well too late in the development cycles.

On top of that, the resulting software is often tailored to the facility that developed the hardware, limiting the re-usability of the hard- and software elsewhere without extensive adaptation.

Our Solution

The solution we are presenting below is tying together several building blocks that have been developed at the Accelerator Beam Controls (MSK) group in the past few years, as well as efforts from the global open source community.

We provide a generic software solution by taking advantage of the interoperability of the DESY FWK [1] with the ChimeraTK [2]. It exposes data to the control system with minor to no configuration effort, using industry standard protocols such OPC UA [3] and EPICS [4]. The data can then easily be consumed by all known scientific control systems or even industry-standard Supervisory Control and Data Acquisition (SCADA) systems, and hardware can be controlled likewise where necessary.

Furthermore, by taking advantage of the on-chip computing power of the new generation FPGAs that integrate a full System on Chip (SoC), it is possible to create a piece of hardware that can be used in quasi-standalone mode and directly plugged into the control system's network environment without having to write any code that runs on an external computer. This will cut down the time-to-machine considerably.

OVERVIEW OF THE BUILDING BLOCKS

The final outcome of this work is not one stand-alone piece of software or configuration. Instead, it consists of several pieces that have already been in development for the past years. The major components of the setup shall be shown below.

Firmware Generation & Hardware Description

The DESY FWK provides the developer with board support packages (BSP), reusable blocks for recurring firmware programming tasks and a unified build environment for the synthesis tools of different FPGA vendors. A complete description of the DESY FWK is beyond the scope of this publication. For further detail on the DESY FWK the reader shall be referred to [5].

One important artifact of the firmware generation process is the register map file. It provides a machine consumable description of hardware addresses to user-readable names and

* The authors acknowledge support from Deutsches Elektronen-Synchrotron DESY Hamburg, Germany, a member of the Helmholtz Association HGF.
† jens.georg@desy.de

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP025

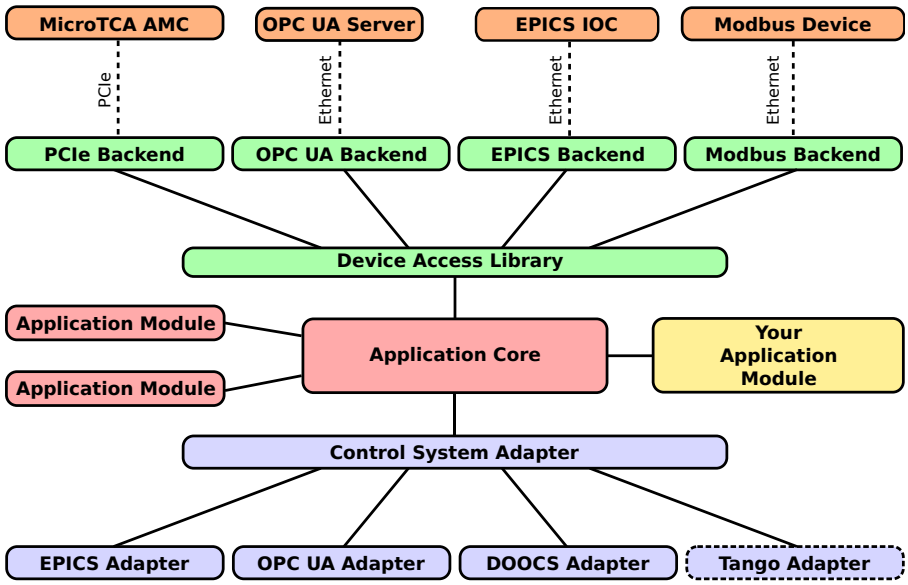

Figure 1: Overview of the ChimeraTK framework.

description of the data transported through those registers. A snippet of such a register file can be seen in Listing 1.

Listing 1: Example for a register map file. It provides names for addresses and details on the data. The columns are the name, the number of elements, the address, the size, an additional address component, information on the interpretation of the bits and the access mode of the register.

```
TIMING.ID              1  0x0080  4  8  32   0   0  RO
TIMING.VERSION         1  0x0084  4  8  32   0   0  RO
TIMING.ENABLE          1  0x0088  4  8   2   0   0  RW
TIMING.SOURCE_SEL      2  0x008C  8  8   8   0   0  RW
TIMING.SYNC_SEL        2  0x0094  8  8   8   0   0  RW
TIMING.DIVIDER_VALUE   2  0x009C  8  8  32   0   0  RW
TIMING.TRIGGER_CNT     2  0x00A4  8  8  32   0   0  RO
TIMING.EXT_TRIGGER_CNT 8  0x00AC 32  8  32   0   0  RO
```

Hardware Interfacing & Control System Access

ChimeraTK is a C++ framework for building data source and control-system agnostic software [6]. Figure 1 shows an overview over the components of ChimeraTK.

At one end of the chain DeviceAccess together with its backends provides the client access. This can be direct by hardware connections such as PCIe devices but also to many networked sources such as Modbus, EPICS or OPC UA. This list is by no means exhaustive. For a full list of supported backends refer to [7, 8].

At the other end, the control system adapter [9] together with its specific adapter implementations, forms the server interface for publishing data to one specific control system middleware. Switching between different control system middlewares can be done without code modification in the user implementation.

ApplicationCore ties client and server interface together. It gives the user the ability to implement small and reusable data processing modules that can be chained to build more complex algorithms.

ApplicationCore also provides robustness against temporary failure of the data sources by providing automatic recovery procedures [10].

Yocto and OpenEmbedded

The Yocto project [11] is a joint effort of multiple hardware vendors under the umbrella of the Linux Foundation [12] to create an extensible framework for building an embedded Linux distribution. It provides the reference distribution Poky, tools for building and extending the distributions, software bill of materials, reproducible builds and license checks. Each Yocto release is accompanied by a curated set of layers from the OpenEmbedded [13] project. A layer contains meta-data and build descriptions for software that logically belongs together. Such a build description for a single piece of software is called a recipe. Figure 2 and the Yocto Project website [14] provide a more in-depth description of its structure.

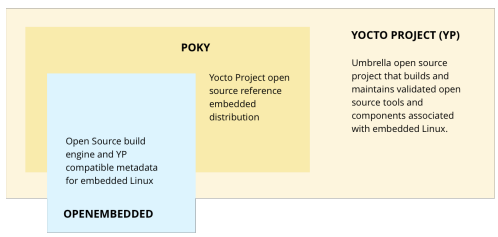

Figure 2: Coarse overview of Yocto's project structure [14].

Many embedded hardware vendors chose to base their BSP on Yocto. AMD Xilinx chose this option also for their SoC platforms and provides the Yocto-based PetaLinux distribution [15].

THE SETUP

Generic Application Servers

At our daily work at MSK we were already confronted with many hardware devices that just needed periodic read-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP025

out of monitored values and configuration parameters, feeding to more complex upstream servers. To ease the repetitive efforts, we have implemented the GenericDeviceServer [16], based on ApplicationCore and DeviceAccess. It uses the configuration facilities that are built into ApplicationCore [17] to expose a set of devices. In its leanest configuration it will expose the complete device to the control system as-is.

If this is not intended, the list of exposed variables can be narrowed further down by using DeviceAccess's logical name mapping [18] and by providing a variable mapper configuration for the chosen control system middleware.

If values from the device need to be transformed before being exposed to the control system, the logical name mapping can be also be used to perform calculations based on variable contents, such as value conversions. This is accomplished using a plug-in for the name mapper. An example for such a value conversion is shown in Listing 2.

Using these mechanisms, many of the basic use-cases of software integration of new hardware can already be accomplished by simple means of configuration for GenericDeviceServer.

Listing 2: Basic example for a thermistor value conversion. We use ExprTk [19] for the expression evaluation. Expression inputs are not limited to the target register but can contain data from other sources as well.

```
<redirectedRegister name="rfModuleTemperature">
    <targetDevice>ULOG_RAW</targetDevice>
    <targetRegister>RF_THERM1_VTG</targetRegister>
    <plugin name="math">
        <parameter name="formula">
        var B := 3940.0;
        var Vd := 5.0;
        var R1 := 51000.0;
        var Ro := 10000.0;
        var To := 298.15;

        return [1.0 / (1.0 / B * log(2 * R1 * x /
    ↪ 10000.0 / ((Vd - x / 10000.0) * Ro)) + 1.0 / To
    ↪ ) - 273.15];
        </parameter>
    </plugin>
</redirectedRegister>
```

Userspace Input/Output (UIO)

While traditional FPGA cards were attached to the CPU using PCI or PCIe, the SoC-based architectures of contemporary FPGAs implement communication channels within the SoC platform itself, e.g. based on Advanced eXtensible Interface (AXI). Custom peripherals that are implemented by the developer in the FPGA logic of such a platform would normally require a custom kernel driver in the Linux operating system. Often one can make use of the UIO [20] framework instead. It provides a generic driver for memory-mapped interfaces to register-based hardware and also basic interrupt support.

We have written a new DeviceAccess backend [21] to access such devices transparently from ChimeraTK. The development of this backend also included a software-controllable UIO dummy Linux kernel driver [22] for use in our unit and conformance test suites [23].

Integration in the Yocto Ecosystem

Through the creation of the ChimeraTK OpenEmbedded layer meta-chimerak [24] we provide to the community a very easy way to integrate our framework into any embedded Linux distribution that is based on Yocto.

Not only do we provide recipes for our core libraries such as DeviceAccess and ApplicationCore, we also include tooling for updating FPGA firmware, Python bindings to DeviceAccess, a graphical device monitor QtHardMon[1] and the aforementioned GenericDeviceServer, together with the OPC UA control system adapter.

The DESY FWK consumes the meta-chimeratk OpenEmbedded layer and integrates it into the firmware build for the SoC-based board. This firmware build also includes a small embedded Linux distribution based on AMD Xilinx PetaLinux. It contains the device register map for the specific configuration of this FPGA at a well-known location inside the root file system of the embedded Linux.

This enables the user to log into the embedded Linux, where one can start the pre-installed GenericDeviceServer and get instant control system integration of the device. The user can also use the included Jupyter Notebook to use the DeviceAcces Python bindings for more low-level interaction or debugging purposes.

Known Limitations

While the use of the GenericDeviceServer might be convenient at first, we acknowledge the necessity for more specific software to be written. However, by using the Yocto-generated Software Development Kit (SDK), developers are put into the position to implement their own ChimeraTK server software directly against the embedded environment.

We also envision situations in which the hardware specifications of the SoC are no longer fitting to the task of the server. If such a situation occurs, it is possible to switch to the traditional model of running the server software on an external CPU, communicating with the hardware through PCIe. This is a minor change in the configuration of the used device backend in the ApplicationCore server. No additional code change is required for this.

EXPERIENCES

The GenericDeviceServer has been deployed at the European XFEL and FLASH facilities, replacing old and unmaintained prototype server software on our Ubuntu setups, sparing the MSK Software team a time-consuming full rewrite of those servers.

The OpenEmbedded layer was already used outside of the DESY FWK in other embedded projects, giving valuable feedback on issues that had been seen with non-64bit CPU architectures [25].

We have provided developers at SOLEIL [26] with early versions of this setup for feedback on our newly designed hardware and firmware. This lead to insights into areas that

[1] optional due to dependency on Qt and X11

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP025

needed improvement for a better end user experience, such as the need for improved documentation for the setup of GenericDeviceServer.

By the time of writing this paper, hardware prototypes containing this setup are given out to internal users at DESY to provide early feedback on the FPGA contained firmware implementation.

CONCLUSION AND OUTLOOK

Through the introduction of a backend for the generic UIO Linux driver interface and the introduction of the OpenEmbedded layer, we have vastly extended the range of devices that can be used with ChimeraTK.

With the GenericDeviceServer and the DESY FWK we are in a position to give users hardware that can integrate into an existing control system environment. It serves as a platform they can jump start their own development, while giving us the needed early feedback. This setup has already been proven to be effective in cutting down development feedback cycle times and finding issues early.

We plan to add further refinement to the OpenEmbedded layers. One part is to conform to the ptest [27] feature provided by Yocto, which integrates our software tests in the distribution's test suite, and also allows the developer to disable the test builds for faster turn-around times during development. The other part is to automatically provide our Application Programming Interface (API) documentation inside the Yocto-SDK.

Currently the bring-up of the GenericDeviceServer on the embedded Linux side still requires some user interaction. We plan to provide simple means of adding configuration and automatic start of the service beforehand, if so desired by the user.

We are also in preparation of adding more control system adapters to our Yocto layer, with Tango being our main priority.

REFERENCES

[1] The DESY Open Source FPGA Framework, https://gitlab.desy.de/fpgafw/fwk

[2] ChimeraTK C++ framework, https://github.com/ChimeraTK/

[3] OpenFoundation OPC UA overview, https://opcfoundation.org/about/opc-technologies/opc-ua/

[4] EPICS, https://epics-controls.org/

[5] L. Butkowski *et al.*, "The DESY Open Source FPGA Framework", presented at ICALEPCS'23, Cape Town, South Africa, Oct 2023, paper MO4AO03, this conference.

[6] M. Killenberg *et. al.*, "Abstracted Hardware and Middleware Access in Control Applications", in *Proc. ICALEPCS'17*, Barcelona, Spain, 2017, paper TUPHA178, pp. 840–845. doi:10.18429/JACoW-ICALEPCS2017-TUPHA178

[7] Backends included in DeviceAccess, https://github.com/ChimeraTK/DeviceAccess/tree/master/device_backends

[8] Third-party DeviceAccess backends, https://github.com/ChimeraTK?q=backend&type=all

[9] M. Killenberg *et al.*, "Integrating control applications into different control systems", in *Proc. ICALEPCS'15*, Melbourne, Australia, 2015, paper TUD3O05, pp. 581–584. doi:10.18429/JACoW-ICALEPCS2015-TUD3O05

[10] M. Killenberg *et al.*, "Automated Device Error Handling in Control Applications", in *Proc. ICALEPCS'21*, Shanghai, Peoples Republic of China, Oct. 2021, pp. 408–412. doi:10.18429/JACoW-ICALEPCS2021-TUPV012

[11] The Yocto Project, https://www.yoctoproject.org/

[12] The Linux Foundation, https://www.linuxfoundation.org/

[13] The OpenEmbedded Project, https://www.openembedded.org/wiki/Main_Page

[14] The Yocto Project Structure, https://www.yoctoproject.org/software-overview/

[15] Xilinx PetaLinux, https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

[16] Generic Device Server, https://github.com/ChimeraTK/GenericDeviceServer

[17] ApplictionCore Server Configuration, https://chimeratk.github.io/ApplicationCore/master/configreader.html

[18] ChimeraTK Logical Name Mapping, https://chimeratk.github.io/DeviceAccess/master/lmap.html

[19] ExprTK, http://www.partow.net/programming/exprtk/

[20] Linux Userspace I/O, https://www.kernel.org/doc/html/latest/driver-api/uio-howto.html

[21] DeviceAccess UIO backend, https://github.com/ChimeraTK/DeviceAccess/tree/master/device_backends/uio

[22] Linux dummy UIO driver, https://github.com/ChimeraTK/uio-dummy/

[23] J. Georg *et al.*, "Continuous Integration and Debian Packaging for Rapidly Evolving Software", presented at ICALEPCS'23, Cape Town, South Africa, Oct 2023, paper MO2BCO07, this conference.

[24] ChimeraTK Yocto Layer, https://github.com/ChimeraTK/meta-chimeratk

[25] Issue report for DeviceAcces on 32 bit platforms, https://github.com/ChimeraTK/DeviceAccess/issues/256

[26] Synchrotron SOLEIL, https://www.synchrotron-soleil.fr/en

[27] Yocto PTEST distro feature, https://wiki.yoctoproject.org/wiki/Ptest