

ChimeraTK.

A software tool kit for hardware access and control application development.



Martin Killenberg
Christian Willner

06th December 2022

11th MicroTCA Workshop
DESY, Hamburg

ChimeraTK — Control system and **H**ardware **I**nterface with **M**apped and **E**xtensible **R**egister-based device **A**bstraction **T**ool **K**it

What does a control application do?

Task 1

- Talk to the hardware
- ChimeraTK DeviceAccess

Task 2

- Run the business logic / algorithm
- ChimeraTK ApplicationCore

Task 3

- Interface to the control system
- ChimeraTK ControlSystemAdapter

ChimeraTK — Control system and Hardware Interface with Mapped and Extensible Register-based device Abstraction Tool Kit

What does a control application do?

Task 1

- Talk to the hardware
- ChimeraTK DeviceAccess

Task 2

- Run the business logic / algorithm
- ChimeraTK ApplicationCore

Task 3

- Interface to the control system
- ChimeraTK ControlSystemAdapter

This tutorial: DeviceAccess

- Take a Struck SIS8300 KU into operation

DeviceAccess.

Talk to the hardware

What do I want to do when accessing hardware?

- Read or write **process variables** which “live” on the hardware
- Talk to components of the device
- Trigger actions

How do I do it?

- Usually data is stored in some **register**
- **Control registers** allow to access components or start actions
- Registers often are addressed with a **numeric scheme**
- There is some **protocol/interface** to access the register space

Each protocol has a different API, but they all provide the same core functionality: Read and write registers.

⇒ Not nice: Lots of **implementation details in your program flow**.

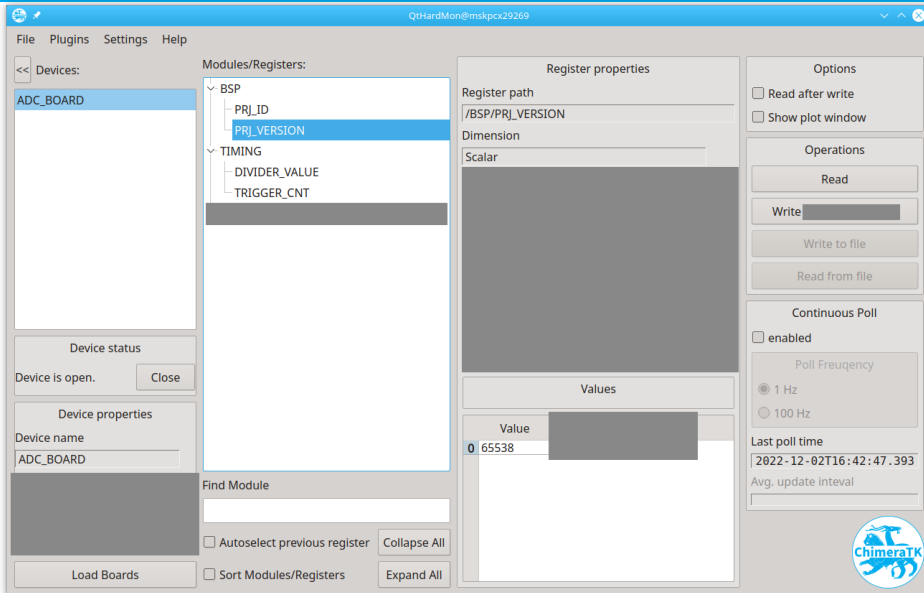
ChimeraTK DeviceAccess

- Focus on the *what*, not on the *how*
- Have one common interface for all protocols and devices

Getting started with DeviceAccess.

A register

- contains **data** (numerical or a string)
- is identified by a **name**
- lives on a **device**
- has a **length** ($1 \hat{=}$ scalar, $> 1 \hat{=}$ array)

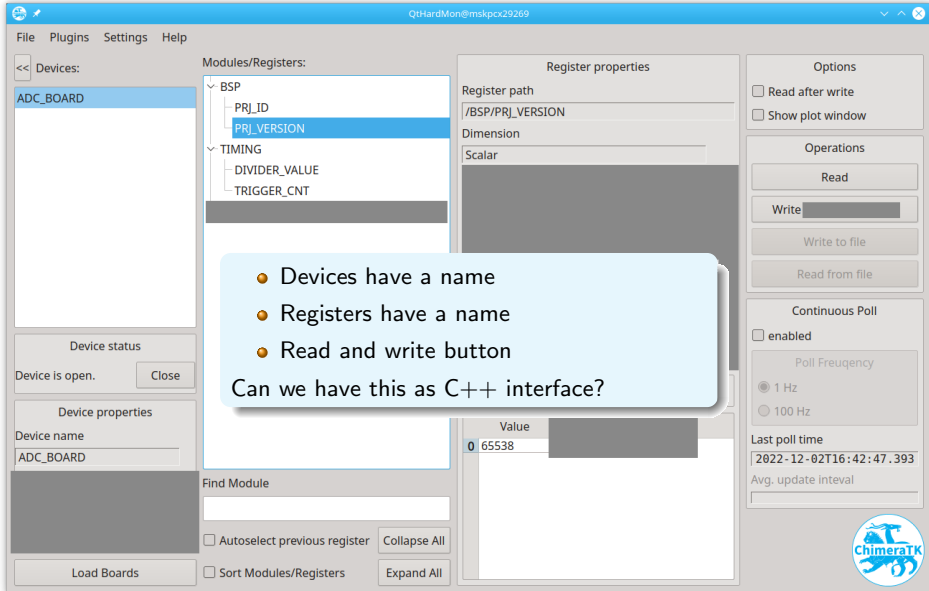


The screenshot displays the QtHardMon application window with the title bar "QtHardMon@mskpcx29269". The interface includes a menu bar (File, Plugins, Settings, Help) and a toolbar. On the left, the "Devices:" panel shows "ADC_BOARD" selected. Below it, the "Device status" section indicates "Device is open." with a "Close" button. The "Device properties" section shows "Device name" as "ADC_BOARD". At the bottom left, there are "Load Boards" and "Autoselect previous register" buttons, along with "Collapse All" and "Expand All" buttons for the register tree.

The central "Modules/Registers:" tree shows a hierarchy: "BSP" (expanded) containing "PRJ_ID" and "PRJ_VERSION" (selected), and "TIMING" containing "DIVIDER_VALUE" and "TRIGGER_CNT". Below the tree is a "Find Module" search bar.

The "Register properties" section on the right shows the "Register path" as "/BSP/PRJ_VERSION", the "Dimension" as "Scalar", and a large "Values" plot area. Below the plot, the "Value" field displays "0 65538".

The right sidebar contains several control panels: "Options" with checkboxes for "Read after write" and "Show plot window"; "Operations" with buttons for "Read", "Write" (with a text input), "Write to file", and "Read from file"; "Continuous Poll" with an "enabled" checkbox, "Poll Frequency" (radio buttons for "1 Hz" and "100 Hz"), "Last poll time" (displaying "2022-12-02T16:42:47.393"), and "Avg. update interval" (with a text input). The "ChimeraTK" logo is visible in the bottom right corner of the sidebar.



The screenshot shows the QtHardMon application window with the title bar "QtHardMon@mskpcx29269". The interface includes a menu bar (File, Plugins, Settings, Help), a "Devices:" list on the left with "ADC_BOARD" selected, and a "Modules/Registers:" tree in the center. The tree shows a hierarchy: BSP (PRJ_ID, PRJ_VERSION) and TIMING (DIVIDER_VALUE, TRIGGER_CNT). The "PRJ_VERSION" register is selected, and its properties are shown in the "Register properties" panel, including the path "/BSP/PRJ_VERSION", dimension "Scalar", and a value of "0 65538". A "Device status" panel at the bottom left indicates "Device is open." with a "Close" button. A "Device properties" panel shows the device name "ADC_BOARD". A "Find Module" section at the bottom has checkboxes for "Autoselect previous register" and "Sort Modules/Registers", and buttons for "Collapse All" and "Expand All". A "Load Boards" button is also present. On the right, there are "Options" (Read after write, Show plot window), "Operations" (Read, Write, Write to file, Read from file), and "Continuous Poll" (enabled, Poll Frequency: 1 Hz, 100 Hz, Last poll time: 2022-12-02T16:42:47.393, Avg. update interval). A "ChimeraTK" logo is in the bottom right corner.

- Devices have a name
- Registers have a name
- Read and write button

Can we have this as C++ interface?

RegisterAccessor

- Has the **data content** of the register
- Has **read()** and **write()** functions to synchronise with the device
- Behaves like a primitive data type (or vector of it) in most use cases

RegisterPath

- Hierarchical register name
- **"/"** as hierarchy separator

Example

ScalarRegisterAccessor<float> behaves like a float with additional read() and write() functions.

```
// Setup section
Device d("ADC_BOARD");
d.open();
auto projectVersion =
    d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

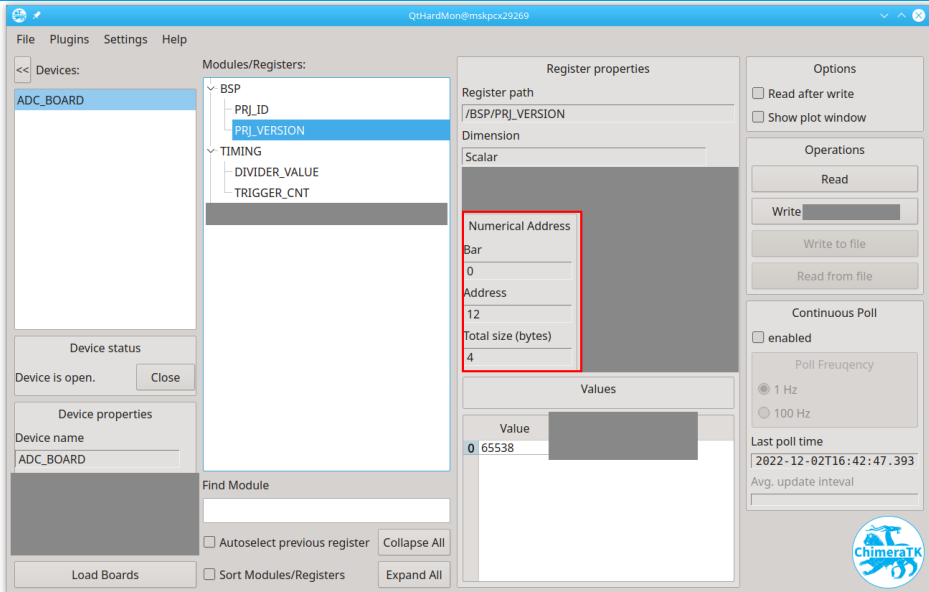
// Business logic
projectVersion.read();
if (projectVersion > 0x010000){
    std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
}
```

- PCI Express identifies registers by address in a "Base Address Range" (BAR)
 - DeviceAccess identifies registers by name
- ⇒ We need a mapping

Example map file

#name	n_words	address	n_bytes	BAR
BSP.PRJ_ID	1	8	4	0
BSP.PRJ_VERSION	1	12	4	0
TIMING.DIVIDER_VALUE	3	8405028	12	0
TIMING.TRIGGER_CNT	3	8405040	12	0

- Map files are automatically created by the DESY (MSK) firmware framework
- Can easily be written manually



The screenshot displays the QtHardMon application window, titled "QtHardMon@mskpcx29269". The interface is divided into several panels:

- Left Panel:** Contains a "Devices:" list with "ADC_BOARD" selected. Below it is a "Device status" section showing "Device is open." with a "Close" button. Further down is a "Device properties" section with "Device name" set to "ADC_BOARD". At the bottom left is a "Load Boards" button.
- Modules/Registers:** A tree view showing the hierarchy: "BSP" (expanded) -> "PRJ_ID" -> "PRJ_VERSION" (selected). Below this is a "Find Module" search bar and checkboxes for "Autoselect previous register" and "Sort Modules/Registers". Buttons for "Collapse All" and "Expand All" are also present.
- Register properties:** A section for the selected register, showing:
 - Register path: "/BSP/PRJ_VERSION"
 - Dimension: "Scalar"
 - Numerical Address: A red box highlights the "Numerical Address" label and the "Bar" field (value 0).
 - Address: "12"
 - Total size (bytes): "4"
- Values:** A table with a single row showing the value "65538" at address "0".
- Options:** Checkboxes for "Read after write" and "Show plot window".
- Operations:** Buttons for "Read", "Write" (with a text input), "Write to file", and "Read from file".
- Continuous Poll:** A section with an "enabled" checkbox, "Poll Frequency" set to "1 Hz", "Last poll time" showing "2022-12-02T16:42:47.393", and an "Avg. update interval" input field.

The ChimeraTK logo is visible in the bottom right corner of the application window.

My device is

- a MicroTCA AMC
- connected via PCI Express

What do I need to access it just as "ADC_BOARD"?

My device is

- a MicroTCA AMC
- connected via PCI Express

What do I need to access it just as "ADC_BOARD"?

- ① A Linux kernel module (driver):
 - Unified MSK PCI Express driver (pcieuni) ✓
 - PCI Express development driver (pciedev) ✓
 - Xilinx XMDA driver ✓

My device is

- a MicroTCA AMC
- connected via PCI Express

What do I need to access it just as “ADC_BOARD”?

- 1 A Linux kernel module (driver):
 - Unified MSK PCI Express driver (pcieuni) ✓
 - PCI Express development driver (pciedev) ✓
 - Xilinx XMDA driver ✓
- 2 A device mapping file with ChimeraTK device descriptor (dmap file)

Example device map file

```
#alias_name      device_descriptor
oven             (xdma:xdma/slot3?map=adc_example.map)
#oven            (dummy?map=adc_example.map)
```

Detour: Working with different backends.

- DeviceAccess supports many different kinds of backends (protocols)
 - Backend can be loaded at link time (recommended) or at run time
 - All backends are accessed with the same interface!
- ⇒ You don't have to learn a new API if you want to talk to a different device.
- Backends are configured through the Chimera Device Descriptor

ChimeraTK Device Descriptor (CDD)

`(backend_type:address?key1=value1&key2=value2)`

Syntax

- Surrounded by parentheses – CDDs can be nested
- `backend_type` – Name of the backend, e.g. "pci", "dummy"
- `address` – Address of the device. The interpretation depends on the backend.
- `keyX=valueX` – List of key-value pairs. The interpretation depends on the backend.

Built-in

pci	PCI-Express (μ TCA AMC)
xdma	PCIe via Xilinx xdma driver (μ TCA AMC)
uio	Backend for linux userspace I/O (FPGA register access on Xilinx SoC) NEW!
rebot	Register based over TCP , lightweight, TPC/IP based inhouse protocol
subdevice	Show part of address space as own logical device
logicalNameMap	Rename and re-organise registers
dummy	Simulate register space in RAM
sharedMemoryDummy	Dummy with address space in shared memory

Loadable plugins

doocs	DOOCS client interface
modbus	Modbus client interface
opu-ua	OPC UA client interface
epics	Native EPICS client interface

Planned

Backends for text-based protocols (e.g. SCPI)

Back to the ADC example...

My device is

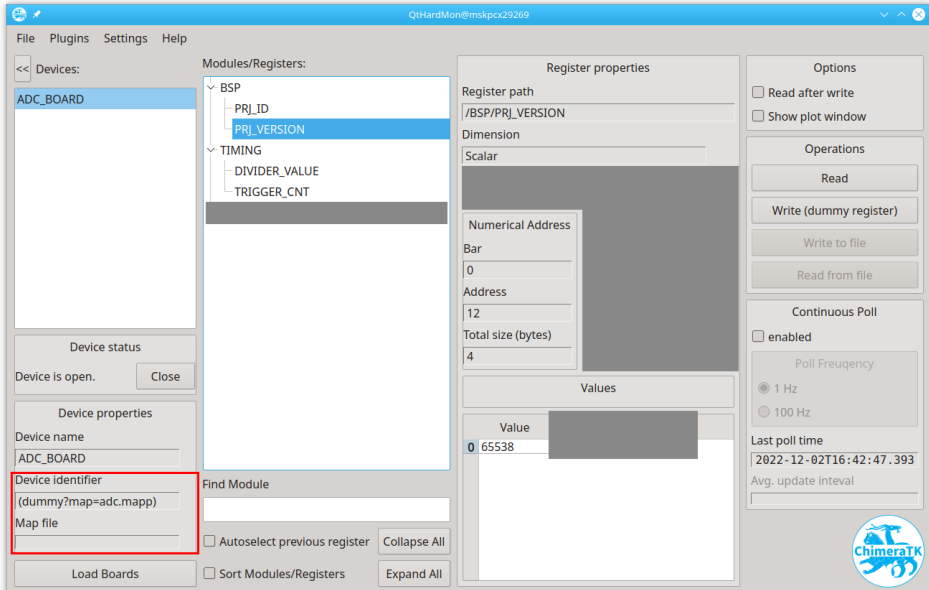
- a MicroTCA AMC
- connected via PCI Express

What do I need to access it just as “ADC_BOARD”?

- 1 A Linux kernel module (driver):
 - Unified MSK PCI Express driver (pcieuni) ✓
 - PCI Express development driver (pciedev) ✓
 - Xilinx XMDA driver ✓
- 2 A device mapping file with ChimeraTK device descriptor (dmap file)

Example device map file

```
#alias_name      device_descriptor
oven             (xdma:xdma/slot3?map=adc_example.map)
#oven            (dummy?map=adc_example.map)
```



The screenshot shows the QtHardMon application window with the title bar "QtHardMon@mskpcx29269". The interface includes a menu bar (File, Plugins, Settings, Help) and a toolbar. The main area is divided into several panels:

- Devices:** A list on the left showing "ADC_BOARD" selected.
- Modules/Registers:** A tree view showing the hierarchy: BSP > PRJ_ID > PRJ_VERSION (selected). Other registers under TIMING include DIVIDER_VALUE and TRIGGER_CNT.
- Register properties:** Fields for Register path (/BSP/PRJ_VERSION), Dimension, and Scalar.
- Options:** Checkboxes for "Read after write" and "Show plot window".
- Operations:** Buttons for "Read", "Write (dummy register)", "Write to file", and "Read from file".
- Continuous Poll:** A checkbox for "enabled", a "Poll Frequency" section with radio buttons for "1 Hz" (selected) and "100 Hz", and a "Last poll time" field showing "2022-12-02T16:42:47.393".
- Device status:** A section indicating "Device is open." with a "Close" button.
- Device properties:** Fields for "Device name" (ADC_BOARD), "Device identifier" (dummy?map=adc.mapp), and "Map file".
- Find Module:** A search bar and buttons for "Autoselect previous register", "Collapse All", "Expand All", and "Sort Modules/Registers".
- Values:** A table showing the current value of the selected register, which is "0 65538".

The "Device identifier" field is highlighted with a red rectangle.

- Transport layer (PCI Express) uses 32 bit words
- Bit interpretation in firmware can be different
 - signed/unsigned integers with less bits (e.g. 25 bits signed)
 - fixend point with fractional bits
 - floating point

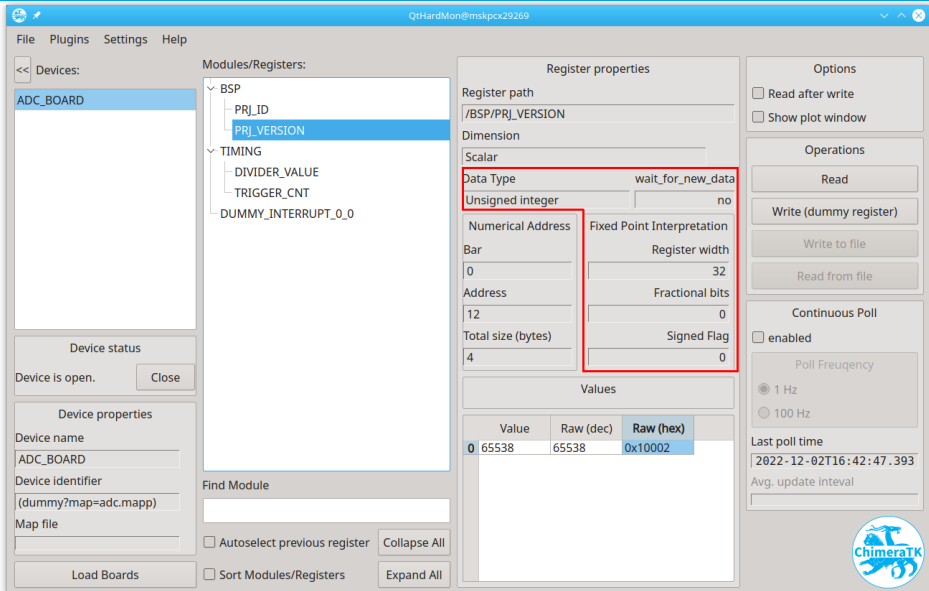
⇒ Extend the mapping with conversion information*

Example map file (excerpt)

#name	n_words	address	n_bytes	BAR	n_bits	bit_interpret	signed	mode
BSP.PRJ_ID	1	8	4	0	32	0	0	RO
BSP.PRJ_VERSION	1	12	4	0	32	0	0	RO
TIMING.DIVIDER_VALUE	3	8405028	12	0	32	0	0	RW
TIMING.TRIGGER_CNT	3	8405040	12	0	32	0	0	INTERRUPT:0

- Readout modes
 - read only (RO)
 - read and write (RW)
 - INTERRUPT (implicitly RO)

* Optional, default conversion is 32 bit signed integer, no fractional bits



The screenshot displays the QtHardMon application window, titled "QtHardMon@mskpcx29269". The interface is divided into several panels:

- Left Panel:** Contains a "Devices:" list with "ADC_BOARD" selected. Below it is a "Device status" section showing "Device is open." with a "Close" button. Further down is a "Device properties" section with fields for "Device name" (ADC_BOARD), "Device identifier" (dummy?map=adc.mapp), and "Map file". At the bottom is a "Load Boards" button.
- Modules/Registers:** A tree view showing a hierarchy: "BSP" (expanded) -> "PRJ_ID" -> "PRJ_VERSION" (selected). Other modules include "TIMING" (with sub-items "DIVIDER_VALUE" and "TRIGGER_CNT") and "DUMMY_INTERRUPT_0_0".
- Register properties:** A detailed view for the selected register. It includes fields for "Register path" (/BSP/PRJ_VERSION), "Dimension" (Scalar), "Data Type" (Unsigned integer), and "wait_for_new_data" (no). A red box highlights the "Data Type" and "wait_for_new_data" fields. Below these are fields for "Numerical Address" (Bar: 0, Address: 12, Total size (bytes): 4) and "Fixed Point Interpretation" (Register width: 32, Fractional bits: 0, Signed Flag: 0).
- Options:** Includes checkboxes for "Read after write" and "Show plot window".
- Operations:** Contains buttons for "Read", "Write (dummy register)", "Write to file", and "Read from file".
- Continuous Poll:** Includes a checkbox for "enabled", a "Poll Frequency" section with radio buttons for "1 Hz" (selected) and "100 Hz", and a "Last poll time" field showing "2022-12-02T16:42:47.393". There is also a field for "Avg. update interval".
- Find Module:** A search bar with buttons for "Autoselect previous register", "Collapse All", "Sort Modules/Registers", and "Expand All".
- Values:** A table showing the current value of the register.

	Value	Raw (dec)	Raw (hex)
0	65538	65538	0x10002

The "ChimeraTK" logo is visible in the bottom right corner of the application window.

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

Complete working example

- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```

Complete working example

- No device/protocol specific code (implementation details)

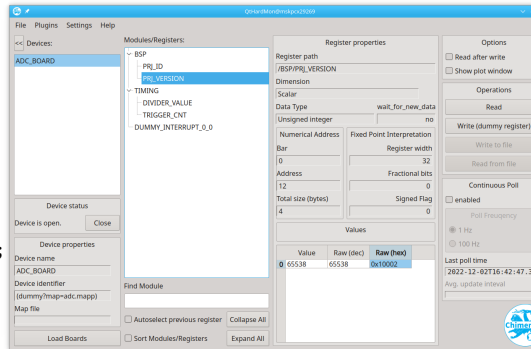
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

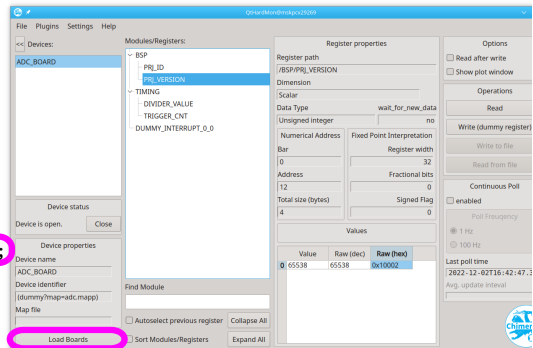
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

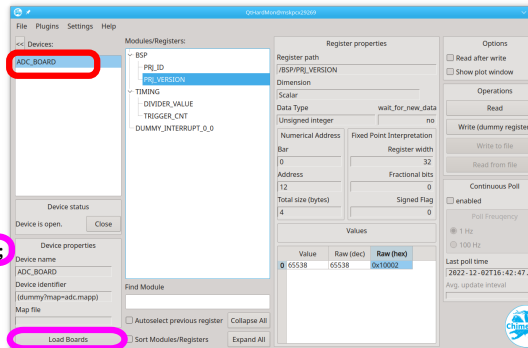
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

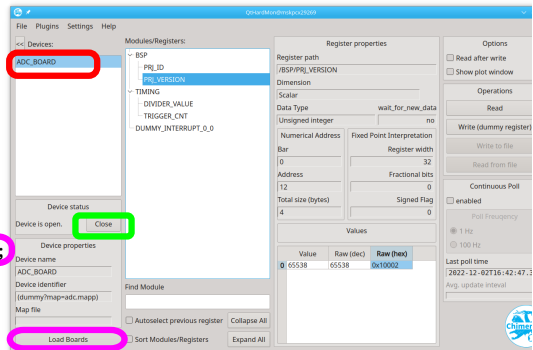
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

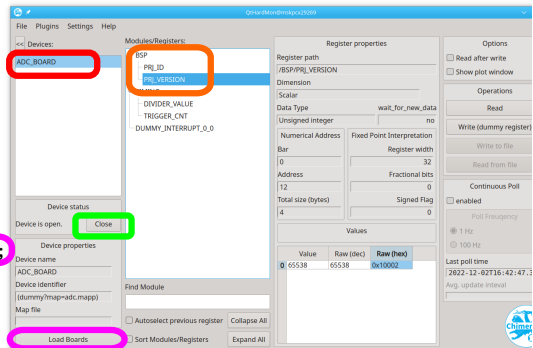
- No device/protocol specific code (implementation details)

```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");
    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

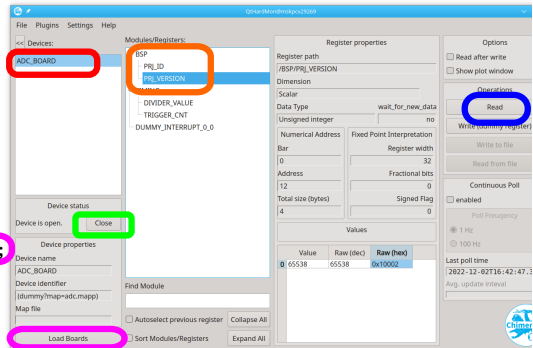
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Complete working example

- No device/protocol specific code (implementation details)

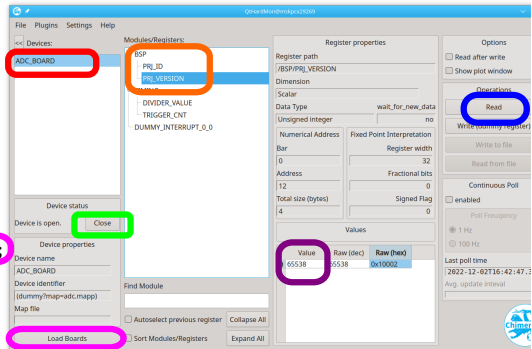
```
#include <iostream>
#include <ChimeraTK/Device.h>

int main(){
    // Setup section
    ChimeraTK::setDMapFilePath("devices.dmap");

    ChimeraTK::Device d("ADC_BOARD");
    d.open();

    auto projectVersion =
        d.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");

    // Business logic
    projectVersion.read();
    if (projectVersion > 0x010000){
        std::cout << "Compatible version " << std::hex << projectVersion << std::endl;
    }
}
```



Installation

- Debian packages for Ubuntu 20.04 are available in the DOOCS Debian repository¹

```
$ sudo apt install libchimeratk-deviceaccess-dev qthardmon
```

Compilation

- Compiler flags are available via pkgconfig

```
$ g++ read_prj_version.cpp -o read_prj_version `pkg-config ChimeraTK-DeviceAccess --cflags --libs`
```

Running the program

```
$ ./read_prj_version  
Compatible version 10002
```

¹Instructions to add the DOOCS debian repository to your system: <https://confluence.desy.de/display/D00CS/D00CS+Installation+Manual>

ScalarRegisterAccessor

- Implicit conversion operator
- ⇒ Automatically converts into the basic data type
- You can do all operations which you can do with the basic data type

```
auto projectVersion =  
    device.getScalarRegisterAccessor<uint32_t>("BSP/PRJ_VERSION");  
  
projectVersion.read();  
if (projectVersion > 0x010000){  
    std::cout << "Compatible version " << std::hex << projectVersion  
                << std::endl;  
    std::cout << "Major version is " << ((projectVersion & 0xFF0000)>>16)  
                << std::endl;  
}
```

OneDRegisterAccessor

- knows the number of elements
- contains data buffer of the correct size
- iterators and [] operator
- can swap() data with std::vector

```
auto triggerCounters
    = device.getOneDRegisterAccessor<uint32_t>("TIMING/TRIGGER_CNT");

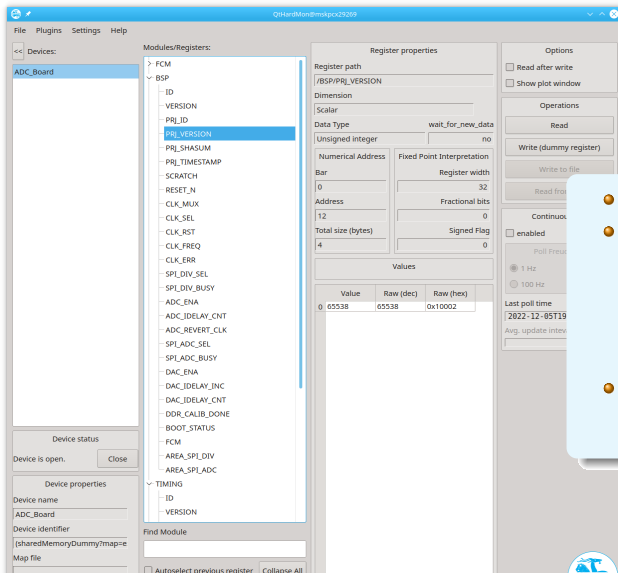
triggerCounters.read();

std::cout << "Trigger counts are ";
for (auto counter : triggerCounters){ // iterate like a std::vector
    std::cout << counter << " ";
}
std::cout << std::endl;
}
```

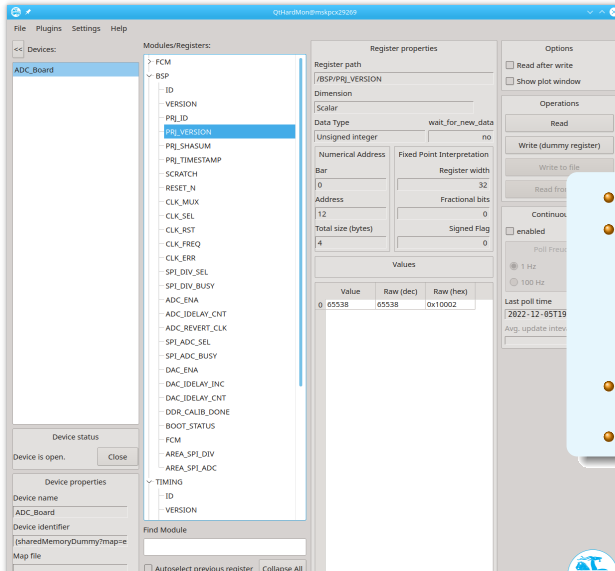
Live demo: Accessing the SIS8300KU.

- ADC board (AMC card)
- MicroTCA crate
- Crate CPU for PCIe connection
- Compatible RTM to get signals via zone 3 connector
- Local frequency generator to synchronise clocks
- ...

We connect to our lab setup for the demo!



- SIS8300 firmware has many registers
- Lots of configuration options
 - Trigger settings
 - Clock settings
 - Sampling frequency
 - ...
- Quite some work to familiarise with it



- SIS8300 firmware has many registers
- Lots of configuration options
 - Trigger settings
 - Clock settings
 - Sampling frequency
 - ...
- Quite some work to familiarise with it
- Can we script the settings?

NEW! Python bindings have a syntax similar to C++

```
import deviceaccess as da
import numpy as np

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()

# For technical reasons ScalarRegisterAccessor inherits from np.ndarray
projectVersion = d.getScalarRegisterAccessor(np.uint32, 'BSP/PRJ_VERSION')

#Business logic
projectVersion.read ();
if projectVersion[0] > 0x010000 :
    print('Compatible version ' + hex(projectVersion[0]))
```

```
import deviceaccess as da
import numpy as np

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()

triggerCounters = d.getOneDRegisterAccessor(np.uint32, 'TIMING/TRIGGER_CNT')

#Business logic
while True:
    triggerCounters.read ();
    print('Trigger ' + str(triggerCounters[2]))
```

```
import deviceaccess as da
import numpy as np

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()

triggerCounters = d.getOneDRegisterAccessor(np.uint32, 'TIMING/TRIGGER_CNT')

#Business logic
while True:
    triggerCounters.read ();
    print('Trigger ' + str(triggerCounters[2]))
```

Issue

This loop is polling with 100 % CPU load!

```
import deviceaccess as da
import numpy as np
import time

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()

triggerCounters = d.getOneDRegisterAccessor(np.uint32, 'TIMING/TRIGGER_CNT')

#Business logic
while True:
    triggerCounters.read ();
    print('Trigger ' + str(triggerCounters[2]))
    time.sleep(0.1)
```

Issue

Asynchronous polling can miss triggers!

AccessMode::wait_for_new_data.

Waiting for data to be pushed by the device

- So far read() operations have polled the hardware (data transfer initialised by the calling code)
- Sometimes you want to wait for data or an event to arrive (push-type operation)
 - Hardware interrupts (PCIe user interrupts)
 - Publish-subscribe protocols like DOOCS ZMQ

AccessMode::wait_for_new_data

- Turn on asynchronous reading in the device
- Specify the access mode `wait_for_new_data` when requesting the accessor
- `read()` will now block until new data has been received^a

^aIf data has already been received before the `read()` call, it returns immediately.

```
triggerCounters = \  
    d.getOneDRegisterAccessor(np.uint32,  
                              'TIMING/TRIGGER_CNT',  
                              accessModeFlags=[da.AccessMode.wait_for_new_data])
```

```
import deviceaccess as da
import numpy as np

# Setup section
da.setDMapFilePath('devices.dmap')
d = da.Device('ADC_BOARD')
d.open ()
d.activateAsyncRead()

triggerCounters = \
    d.getOneDRegisterAccessor(np.uint32, 'TIMING/TRIGGER_CNT',
                              accessModeFlags=[da.AccessMode.wait_for_new_data])

# This handshake register will be integrated into the backend in future
handshake = d.getVoidRegisterAccessor('BSP/IRQ_CLEAR')

#Business logic
while True:
    triggerCounters.read ();
    handshake.write();

    print('Trigger ' + str(triggerCounters[2]))
```

Arrange the register content to logically match your application

- Rename registers
- Add constant registers or dummy registers
- Extract channels from 2D registers and give it a name
- Extract scalars from 1D registers and give it a name
- Extract bits from a scalar register

Math plugin

- Apply conversion formulas, e.g. ADC counts to physical units

Abstract away connection details

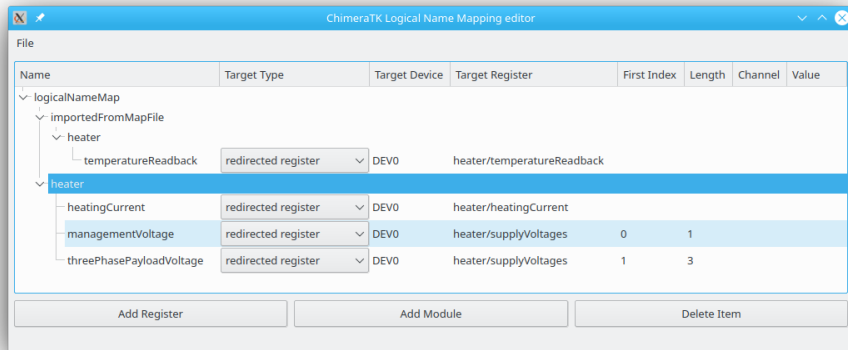
- The oven has a management voltage and 3 phase payload power
- All voltages are recorded on a 4 channel ADC: supplyVoltages

Logical name mapping

supplyVoltages[0]	→	managementVoltage
supplyVoltages[1-3]	→	threePhasePayloadVoltage

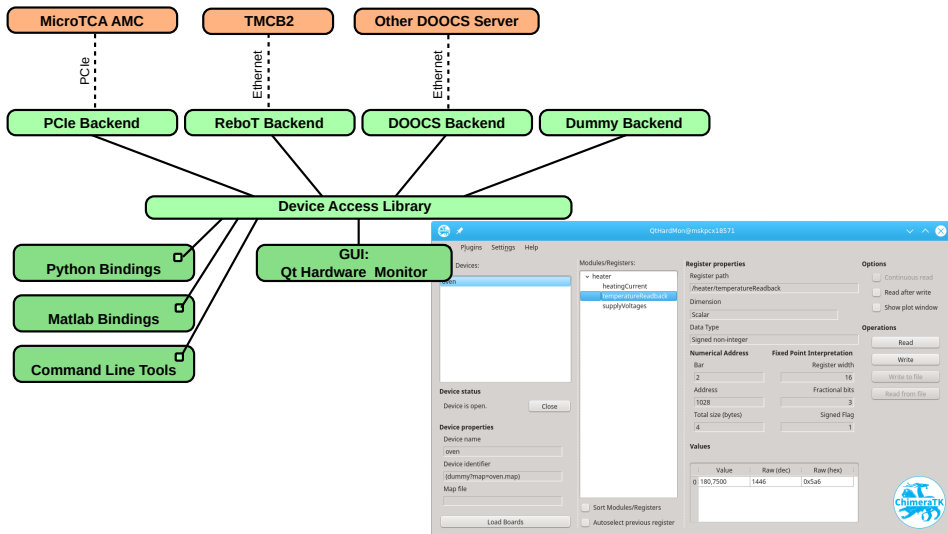
Different deployment options

- Different temperature sensors can be connected
 - Use math plugin with correct formula to convert ADC counts to temperature
- Logical name mapping brings flexibility
 - New sensors can just be configured without re-compiling the server



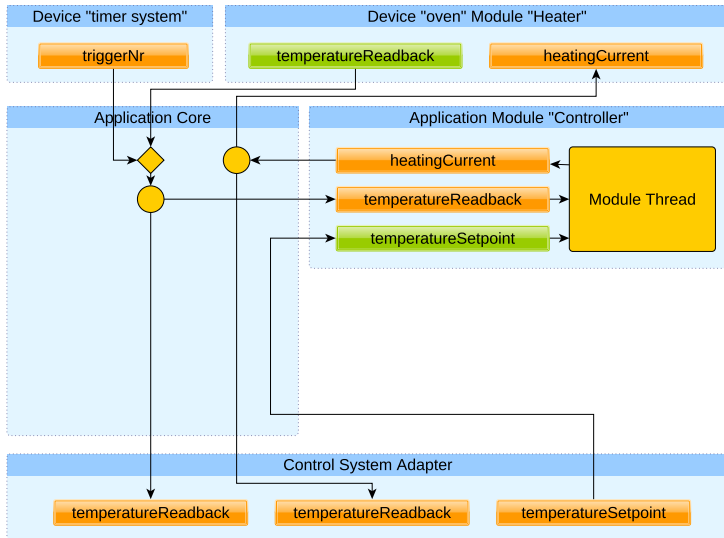
- Import map file as starting point
- Drag and drop registers into modules
- Rename/re-arrange registers
- Planned: Add plugins (currently have to be added manually in the xml file)

Tool under development.
Please give feedback or help implementing missing features.



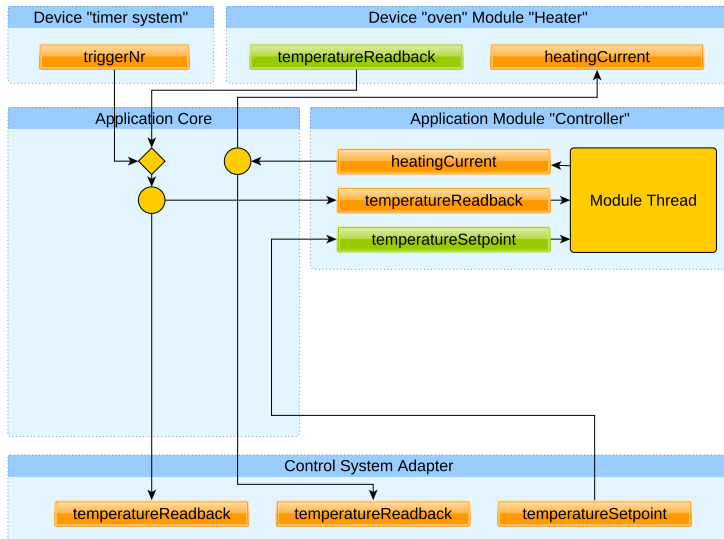
ApplicationCore and ControlSystemAdapter.

Implement the control algorithm



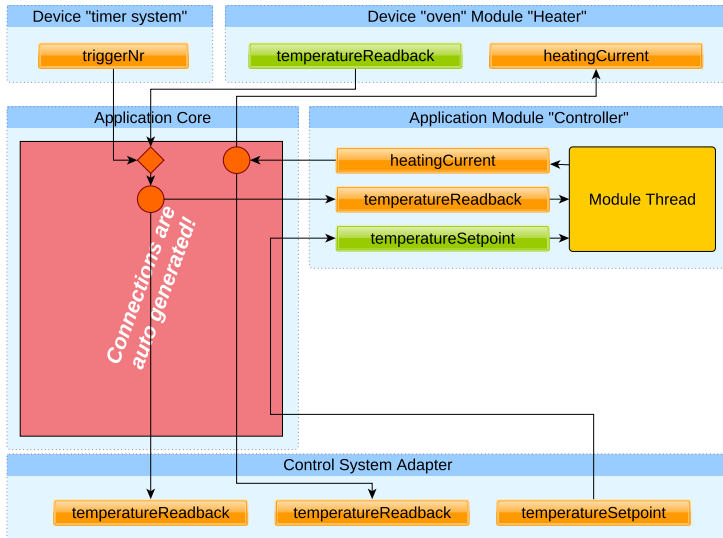
Modules

- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter



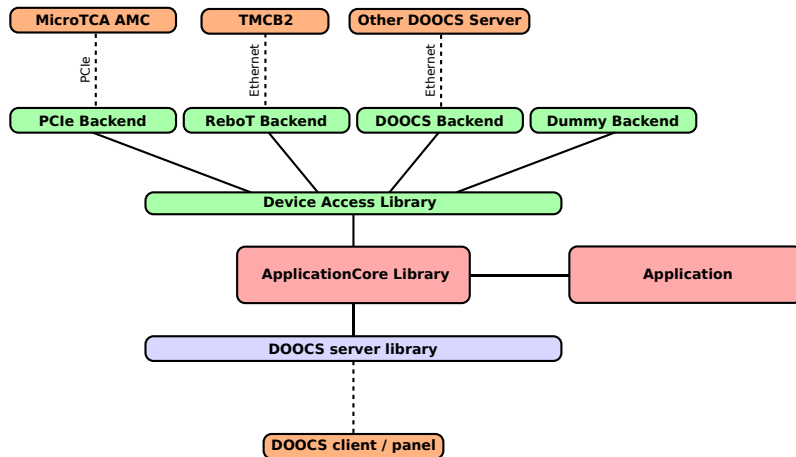
Modules

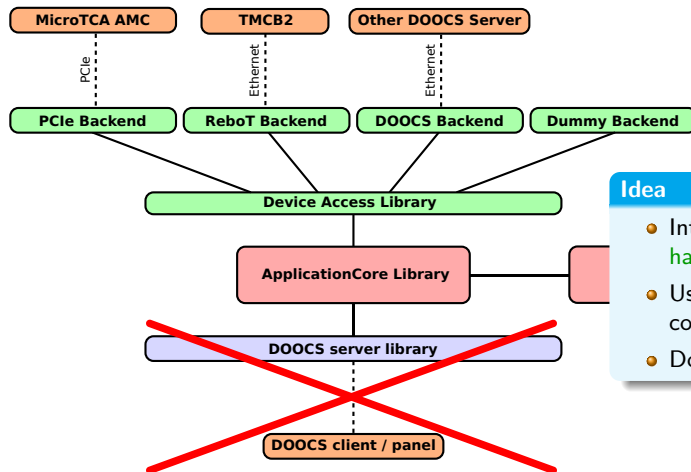
- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter
- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained
- Modern multithreading: lock-free communication between modules ("for free")



Modules

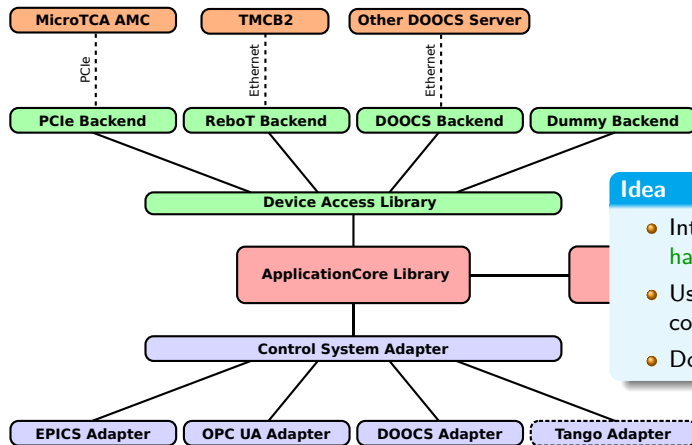
- Input/output variables
- Application Modules
 - One thread per module
- Special "modules"
 - Device module
 - Control system adapter
- Algorithms don't need to know how variables are connected
- Perfect modularity, as modules are self-contained
- Modern multithreading: lock-free communication between modules ("for free")
- Connections are auto-generated





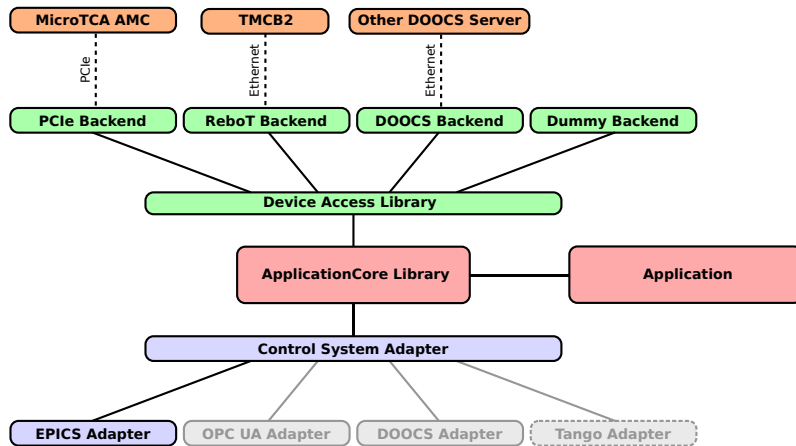
Idea

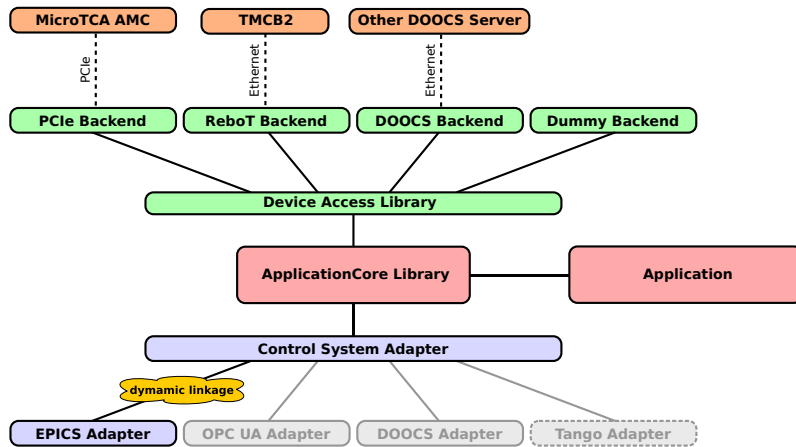
- Introduce similar abstraction as for the **hardware access**
- Use ChimeraTK applications with various control system middlewares
- Do **not** create a new control system!

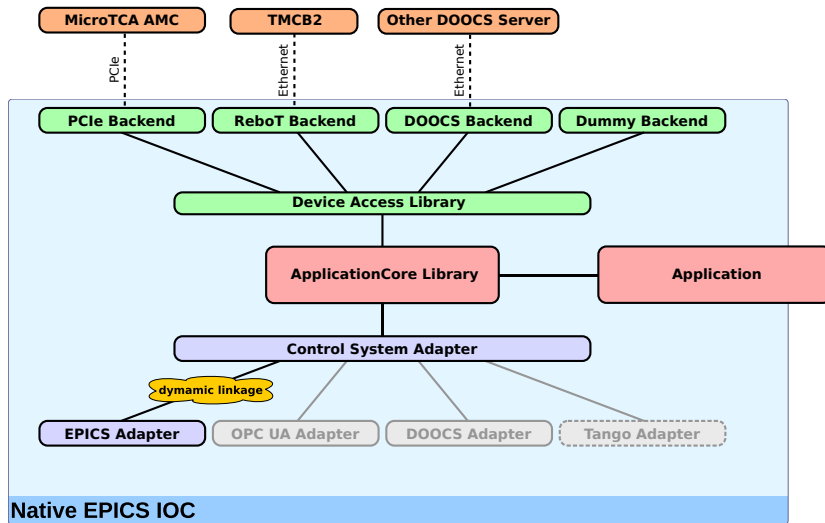


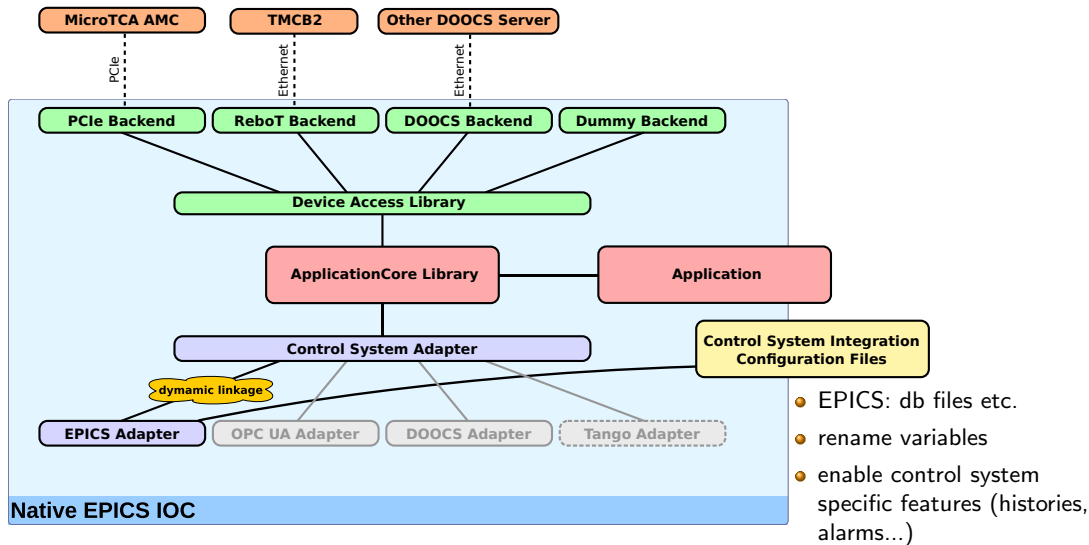
Idea

- Introduce similar abstraction as for the **hardware access**
- Use ChimeraTK applications with various control system middlewares
- Do **not** create a new control system!









ApplicationCore

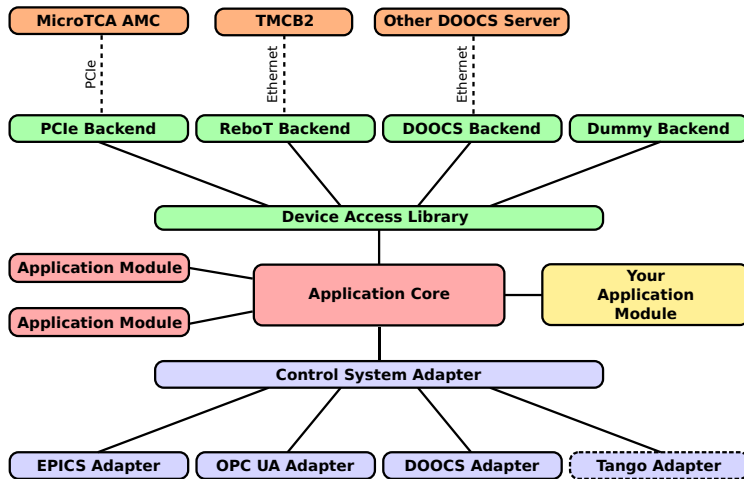
- Automatic device exception handling
- Hierarchy modifiers
- Status monitors and aggregators
- Bi-directional variables

DeviceAccess

- Logical name mapping
- Synchronising several registers (for instance on the macro pulse number)
- Waiting for *any* push-type accessor
- 2D arrays
- Accessing multiplexed data

Under development

- ControlSystemAdapter TangoAdapter (by SOLEIL,
see Status of FOFB upgrade at SOLEIL by Romain Broucqart, Wednesday Session 5)
- We have only implemented what we needed so far
 - Let us know which features you would like to see!





Software Repositories

All software is published under the GNU GPL or the GNU LGPL.

- ChimeraTK source code: <https://github.com/ChimeraTK>
- Ubuntu 20.04 packages are available in the [DESY DOOCS repository](#).

Documentation and Tutorials

- API documentation <https://chimeratk.github.io/>
- Tutorials on the [MicroTCA Workshop Indico page](#)
- e-mail support: chimeratk-support@desy.de

Backup.

`dummy`

- Simulate the I/O address space in RAM
- Information taken from the map file
- Used for unit tests inside one executable

`ExceptionDummy`²

- Controlled way to raise exceptions
- Test the exception handling

`sharedMemoryDummy`

- Multiple processes can interact with the same dummy device

Examples

- Use QtHardMon to interact with a running server
- Write Python script to simulate firmware behaviour
- Automated test of the complete server executable

²Currently a separate backend. Will be integrated into `dummy`.

Logical Name Mapping Example.

Abstracting firmware implementation details from the server

Firmware has different abstraction level than the server

- Closer to the hardware
- More implementation details

A more realistic map file

# name	n_elements	address	size	bar	width	fracbits	signed	R/W
APP.0.TEMPERATURE_ADC	1	0	4	0	16	0	0	RO
APP.0.HEATING_CURRENT	1	4	4	0	32	16	0	RW
APP.0.SUPPLY_ADCS	4	8	16	0	16	0	0	RO

- APP.0 instead of heater
 - ADC for temperature
- ⇒ Conversion formula depends on connected sensor
- ADCs for supply voltages
- ⇒ Need to convert to volts

Firmware has different abstraction level than the server

- Closer to the hardware
- More implementation details

A more realistic map file

#	name	n_elements	address	size	bar	width	fracbits	signed	R/W
APP.0.	TEMPERATURE_ADC	1	0	4	0	16	0	0	RO
APP.0.	HEATING_CURRENT	1	4	4	0	32	16	0	RW
APP.0.	SUPPLY_ADCS	4	8	16	0	16	0	0	RO
APP.0.	POWER	1	24	4	0	1	0	0	RW
APP.0.	SUPPLY_ADCS_GAIN	4	28	16	0	32	0	0	RW
APP.0.	AREA_FW_UPGRADE	1024	256	4096	0	32	0	0	RW

- APP.0 instead of heater
- ADC for temperature
- ⇒ Conversion formula depends on connected sensor
- ADCs for supply voltages
- ⇒ Need to convert to volts
- Register to turn on the power
- Adjustable gain for amplifiers
- AREA_FW_UPGRADE: **Better not touch!**

Device mapping file

```
oven_raw  (sharedMemoryDummy?map=oven.map)  
oven      (logicalNameMap?map=oven.xlmap)
```

- Device oven is now the logical name mapping device
- It uses the hardware accessing device oven_raw

```
<logicalNameMap>
  <module name="heater">

    <redirectedRegister name="heatingCurrent">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/HEATING_CURRENT</targetRegister>
    </redirectedRegister>

    ...
  </module>

  ...
</logicalNameMap>
```

```
<logicalNameMap>
  <module name="heater">

    <redirectedRegister name="heatingCurrent">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/HEATING_CURRENT</targetRegister>
    </redirectedRegister>

    <redirectedRegister name="temperatureReadback">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/TEMPERATURE_ADC</targetRegister>

    </redirectedRegister>

    ...
  </module>

  ...
</logicalNameMap>
```

```
<logicalNameMap>
  <module name="heater">

    <redirectedRegister name="heatingCurrent">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/HEATING_CURRENT</targetRegister>
    </redirectedRegister>

    <redirectedRegister name="temperatureReadback">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/TEMPERATURE_ADC</targetRegister>

      <plugin name="math">
        <!-- 16 bit ADC with 10 V, Sensor: 65 V/degC -->
        <parameter name="formula">x / 2^16 * 10 * 65</parameter>
      </plugin>
    </redirectedRegister>

    ...
  </module>

  ...
</logicalNameMap>
```

```
<logicalNameMap>
  <module name="heater">
    ...
    <redirectedRegister name="managementVoltage">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/SUPPLY_ADCS</targetRegister>
      <numberOfElements>1</numberOfElements>
      <plugin name="math"> <!-- 16 bit ADC with 32 V -->
        <parameter name="formula"> $x / 2.^{16} * 32.$ </parameter>
      </plugin>
    </redirectedRegister>
```

```
    ...
  </module>
  ...
</logicalNameMap>
```

```
<logicalNameMap>
  <module name="heater">
    ...
    <redirectedRegister name="managementVoltage">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/SUPPLY_ADCS</targetRegister>
      <numberOfElements>1</numberOfElements>
      <plugin name="math"> <!-- 16 bit ADC with 32 V -->
        <parameter name="formula"> $x / 2.^{16} * 32.$ </parameter>
      </plugin>
    </redirectedRegister>

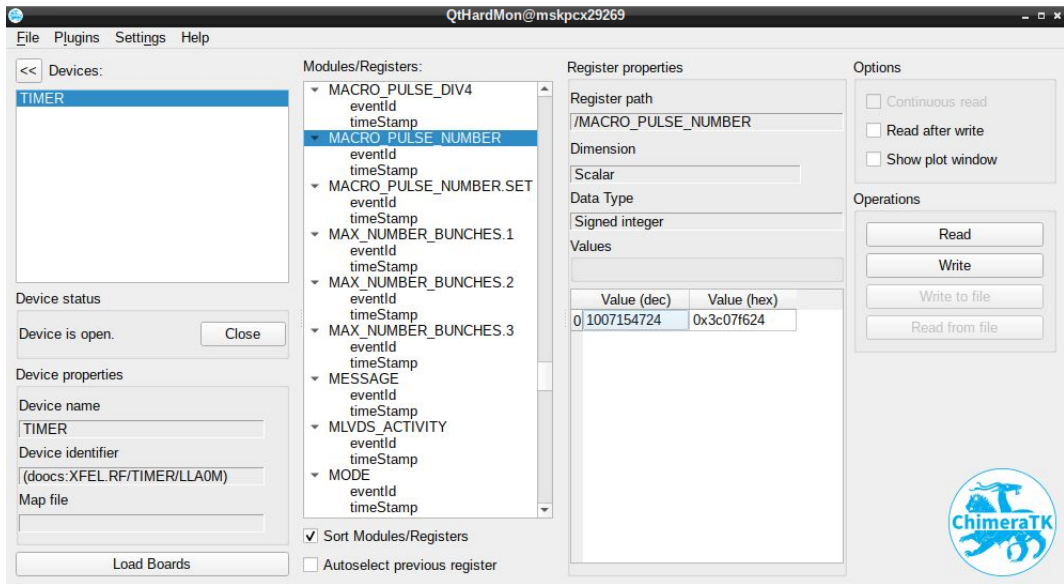
    <redirectedRegister name="threePhaseVoltages">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/SUPPLY_ADCS</targetRegister>\
      <targetStartIndex>1</targetStartIndex>
      <numberOfElements>3</numberOfElements>
      <plugin name="math"> <!-- 16 bit ADC with 650 V (array syntax)-->
        <parameter name="formula">return[x / 2.^16 * 650.]</parameter>
      </plugin>
    </redirectedRegister>

    ...
  </module>
  ...
</logicalNameMap>
```

```
<logicalNameMap>
  <module name="heater">
    ...
  </module>

  <module name="settings">
    <redirectedRegister name="supplyVoltageAdcGains">
      <targetDevice>oven_raw</targetDevice>
      <targetRegister>APP/0/SUPPLY_ADCS_GAIN</targetRegister>
    </redirectedRegister>
  </module>
</logicalNameMap>
```

- Map ADC gains register into config module
- Only heater is used in the server → ADC gains will not show up in the CS
- AREA_FW_UPGRADE is not mapped at all



The screenshot shows the QtHardMon application window with the title bar "QtHardMon@mskpcx29269". The interface is divided into several panels:

- File Plugins Settings Help**: The top menu bar.
- Devices:** A list on the left with "TIMER" selected.
- Device status**: A section below the devices list showing "Device is open." with a "Close" button.
- Device properties**: A section with fields for "Device name" (TIMER), "Device identifier" ((doocs:XFEL.RF/TIMER/LLA0M)), and "Map file" (empty).
- Load Boards**: A button at the bottom of the device properties section.
- Modules/Registers:** A tree view in the center showing a hierarchy of modules and registers. The "MACRO_PULSE_NUMBER" register is selected.
- Register properties**: A panel on the right showing the selected register's details.
 - Register path**: /MACRO_PULSE_NUMBER
 - Dimension**: Scalar
 - Data Type**: Signed integer
 - Values**: A table showing the current value in decimal and hexadecimal.
- Options**: A panel on the right with checkboxes for "Continuous read", "Read after write", and "Show plot window".
- Operations**: A panel on the right with buttons for "Read", "Write", "Write to file", and "Read from file".

Modules/Registers:

- MACRO_PULSE_DIV4
 - eventId
 - timeStamp
- MACRO_PULSE_NUMBER (selected)
 - eventId
 - timeStamp
- MACRO_PULSE_NUMBER.SET
 - eventId
 - timeStamp
- MAX_NUMBER_BUNCHES.1
 - eventId
 - timeStamp
- MAX_NUMBER_BUNCHES.2
 - eventId
 - timeStamp
- MAX_NUMBER_BUNCHES.3
 - eventId
 - timeStamp
- MESSAGE
 - eventId
 - timeStamp
- MLVDS_ACTIVITY
 - eventId
 - timeStamp
- MODE
 - eventId
 - timeStamp

☒ Sort Modules/Registers
☐ Autoselect previous register

Register properties:

Register path: /MACRO_PULSE_NUMBER

Dimension: Scalar

Data Type: Signed integer

Values:

Value (dec)	Value (hex)
0	0x3c07f624

Options:

- ☐ Continuous read
- ☐ Read after write
- ☐ Show plot window

Operations:

Read
Write
Write to file
Read from file

ChimeraTK logo

Loading a DoocsBackend in the dmap file, so QtHardMon knows about it:

```
@LOAD_LIB /usr/lib/libChimeraTK-DeviceAccess-DoocsBackend.so.01.00 focal2  
TIMER (doocs:XFEL.RF/TIMER/LLA0M)
```

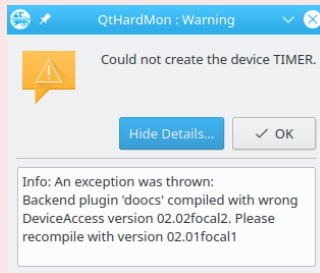
Loading a DoocsBackend in the dmap file, so QtHardMon knows about it:

```
@LOAD_LIB /usr/lib/libChimeraTK-DeviceAccess-DoocsBackend.so.01.00focal2  
TIMER (doocs:XFEL.RF/TIMER/LLA0M)
```

Disclaimer

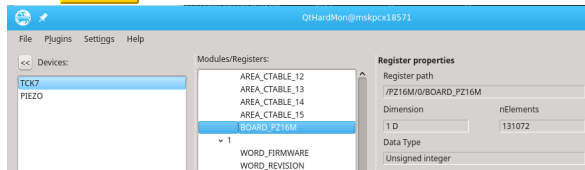
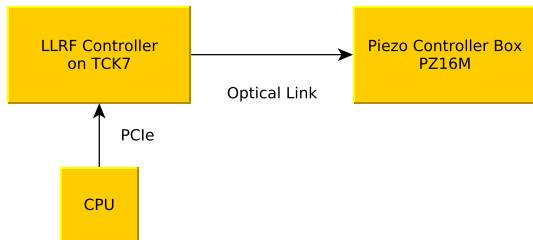
You need the DoocsBackend to be compiled with the **exact** same version of DeviceAccess that was used to compile QtHardMon,

otherwise



Recommended: [Link your executable to the required backends at compile time!](#)

(Run time loading needed for generic tools like QtHardMon, which should be DOOCS-independent.)



- Firmware maps the register space of a subdevice (PIEZO) into a 1D register of its own address space (TCK7). (Usually offset address in a numerically addressed space).
- Both devices have firmwares which evolve separately.
- The Subdevice backend allows to access the subdevice through the parent device as a separate logical entity.

⇒ Separate map file to describe the subdevice.

#alias device_descriptor

TCK7 (pci:llrfutcs4?map=llrf_ctrl_tck7b_acc1_r2097.map)

PIEZO (subdevice:area,TCK7,PZ16M.0.BOARD_PZ16M?map=piezo_pz16m_acc1_r2323.map)

Direct Memory Access

- Efficient way to transfer large data blocks
 - No CPU load because the device directly writes to RAM
 - No space needed in the PCIe I/O address space
- Special ioctl call on the driver

Direct Memory Access

- Efficient way to transfer large data blocks
 - No CPU load because the device directly writes to RAM
 - No space needed in the PCIe I/O address space
- Special ioctl call on the driver

Trick in DeviceAccess

- PCIe has 6 base address ranges (BARs 0 to 5)
- Introduce a new address range 13 in the map file
- The backend knows what to do for this special “BAR”

Map file with 4 MB DMA area

#name	n_words	address	n_bytes	BAR
area_dma	0x100000	0	0x400000	13

Direct Memory Access

- Efficient way to transfer large data blocks
 - No CPU load because the device directly writes to RAM
 - No space needed in the PCIe I/O address space
- Special ioctl call on the driver

Trick in DeviceAccess

- PCIe has 6 base address ranges (BARs 0 to 5)
- Introduce a new address range 13 in the map file
- The backend knows what to do for this special “BAR”

Map file with 4 MB DMA area

#name	n_words	address	n_bytes	BAR
area_dma	0x100000	0	0x400000	13

What changes in the API?

- Nothing! Just use a OneDRegisterAccessor

```
auto myBigDataBlock = d.getOneDRegisterAccessor<int32_t>("area_dma");
```