ChimeraTK Software Framework

Getting started in device server development

Dietrich Rothe, Martin Killenberg, DESY – MSK

Dec 5 2023

12th MicroTCA Workshop for Industry and Research

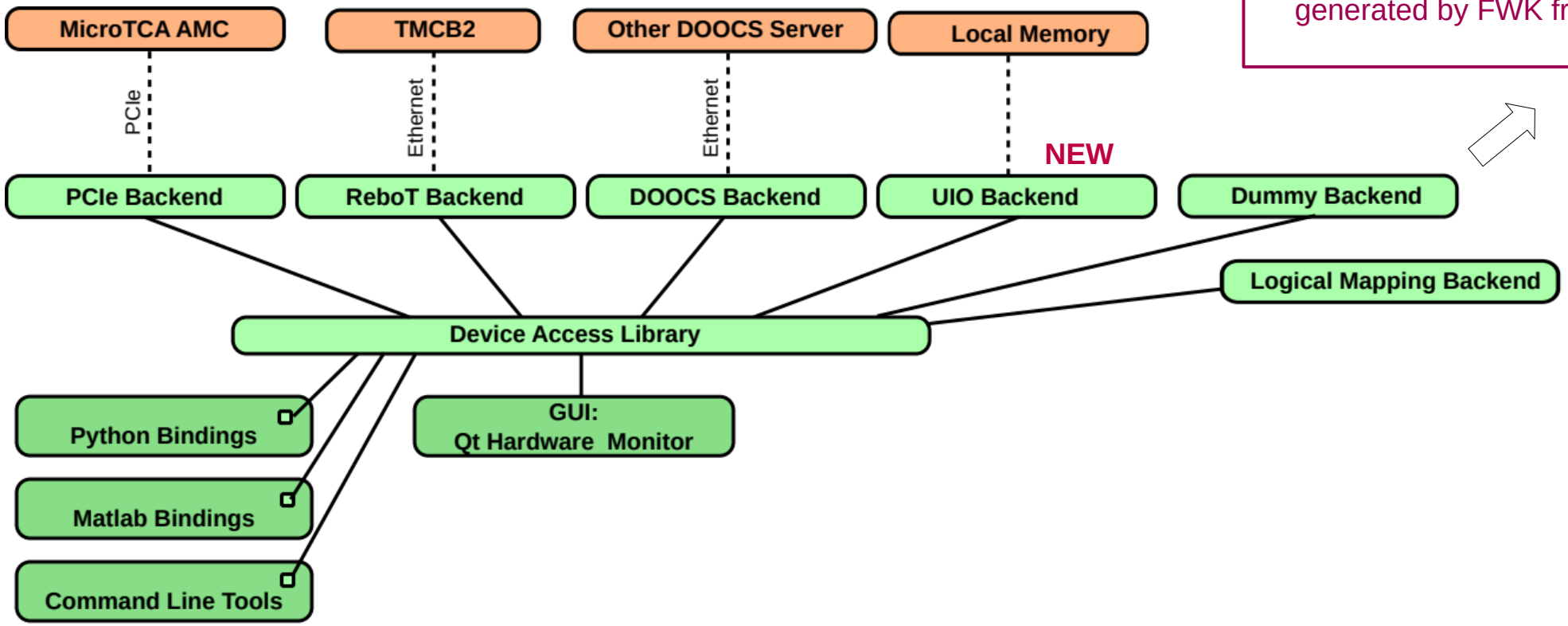
DESY, Hamburg

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

ChimeraTK Framework Overview

DeviceAccess

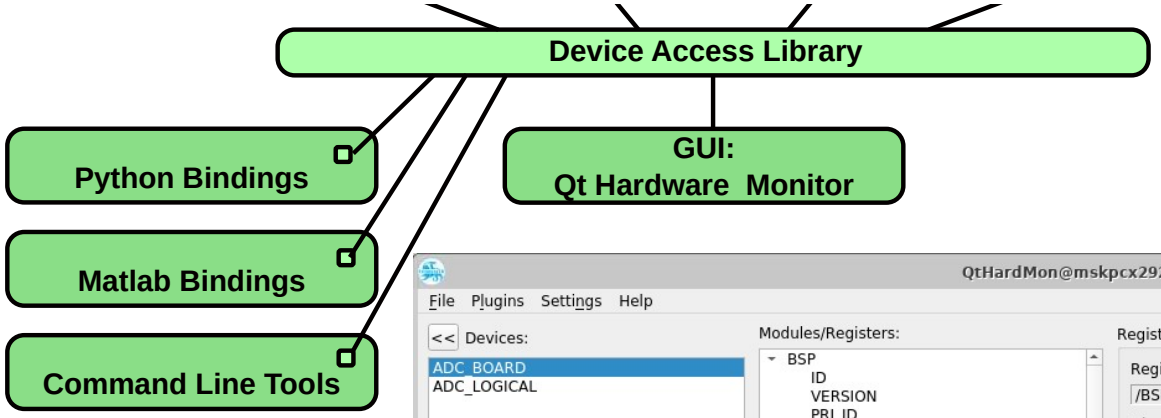
Register Catalogue
- Abstract register path names “Module/RegX”
- e.g. map – file for PCIE register addresses generated by FWK framework



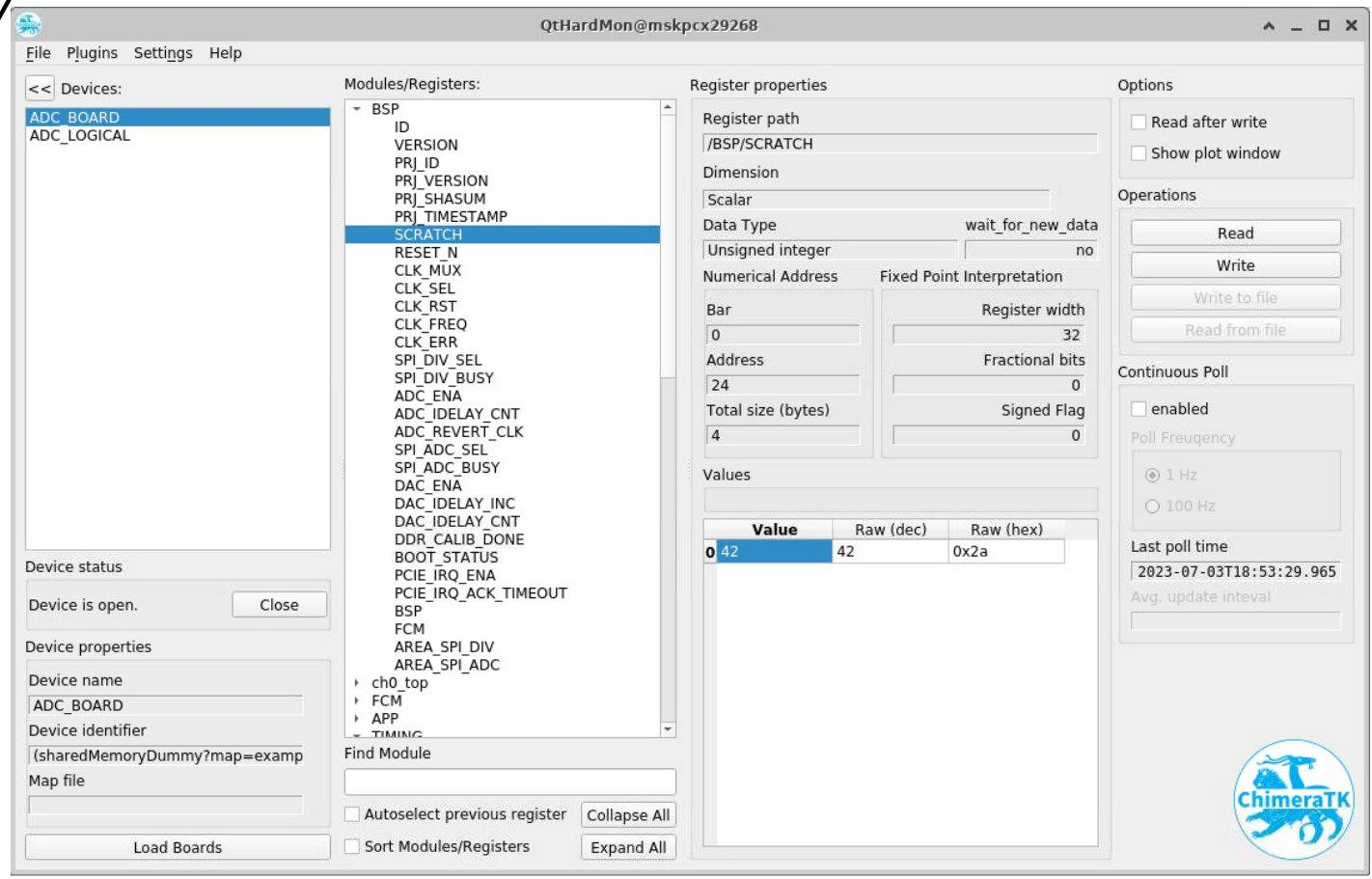
ChimeraTK Framework Overview

DeviceAccess

Accessor
- data buffer: use like a variable
- read(), write(): sync with hardware




```
# initscript.py
import deviceaccess as da
import numpy as np

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

acc = d.getOneDRegisterAccessor(np.uint32,
        'TIMING/DIVIDER_VALUE')
acc[0] = 12499999
acc.write()

...
```



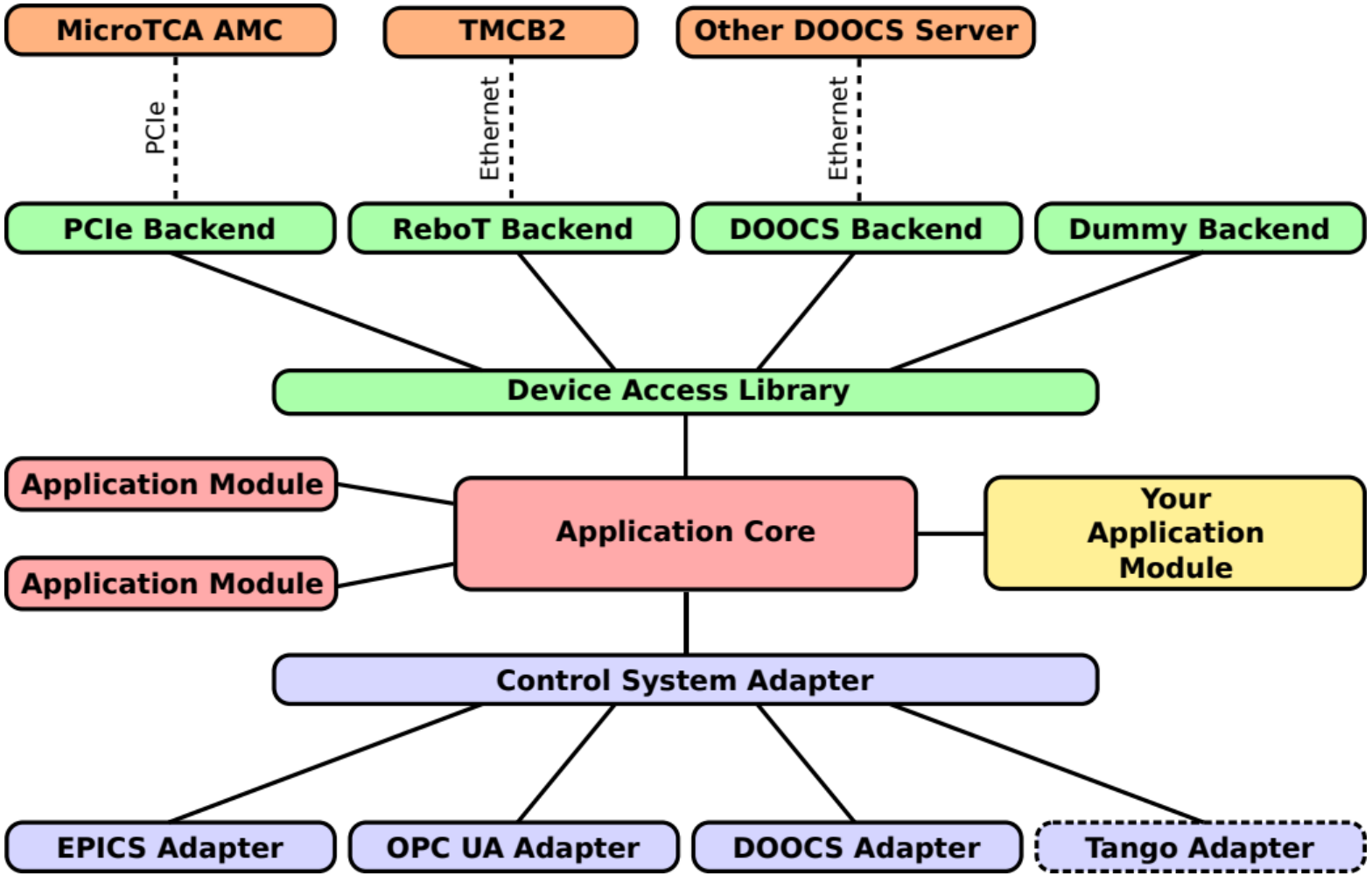
ChimeraTK Framework Overview

DeviceAccess backend list

Hardware access	
pcieuni	PCI-Express via pcieuni driver
xdma	PCI-Express via Xilinx xDMA driver **NEW**
rebot	Register based over TCP = lightweight inhouse protocol
Clients	
doocs	DOOCS client interface
opcua	OPC UA client interface
epics	Native EPICS client interface
modbus	Modbus client interface
Plugins and dummies	
logicalNameMap	Rename and re-organize registers; various feature plugins
subdevice	Show part of address space as own logical device
dummy	Simulate register space in RAM
sharedMemoryDummy	Simulate register space in shared memory
ExceptionDummy	Test error handling (automated tests)

ChimeraTK Framework Overview

Ecosystem



First Board Setup

- Add raw devices
  - List devices and access protocols $\Longrightarrow$ devices.dmap
  - Register set: *.map
  - Initialization script!
    Use DeviceAccess Python bindings

```
# initscript.py
import deviceaccess as da
import numpy as np

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

acc = d.getOneDRegisterAccessor(np.uint32,
        'TIMING/DIVIDER_VALUE')
acc[0] = 12499999
acc.write()

...
```

(**C**himeraTK **D**evice **D**escriptor)

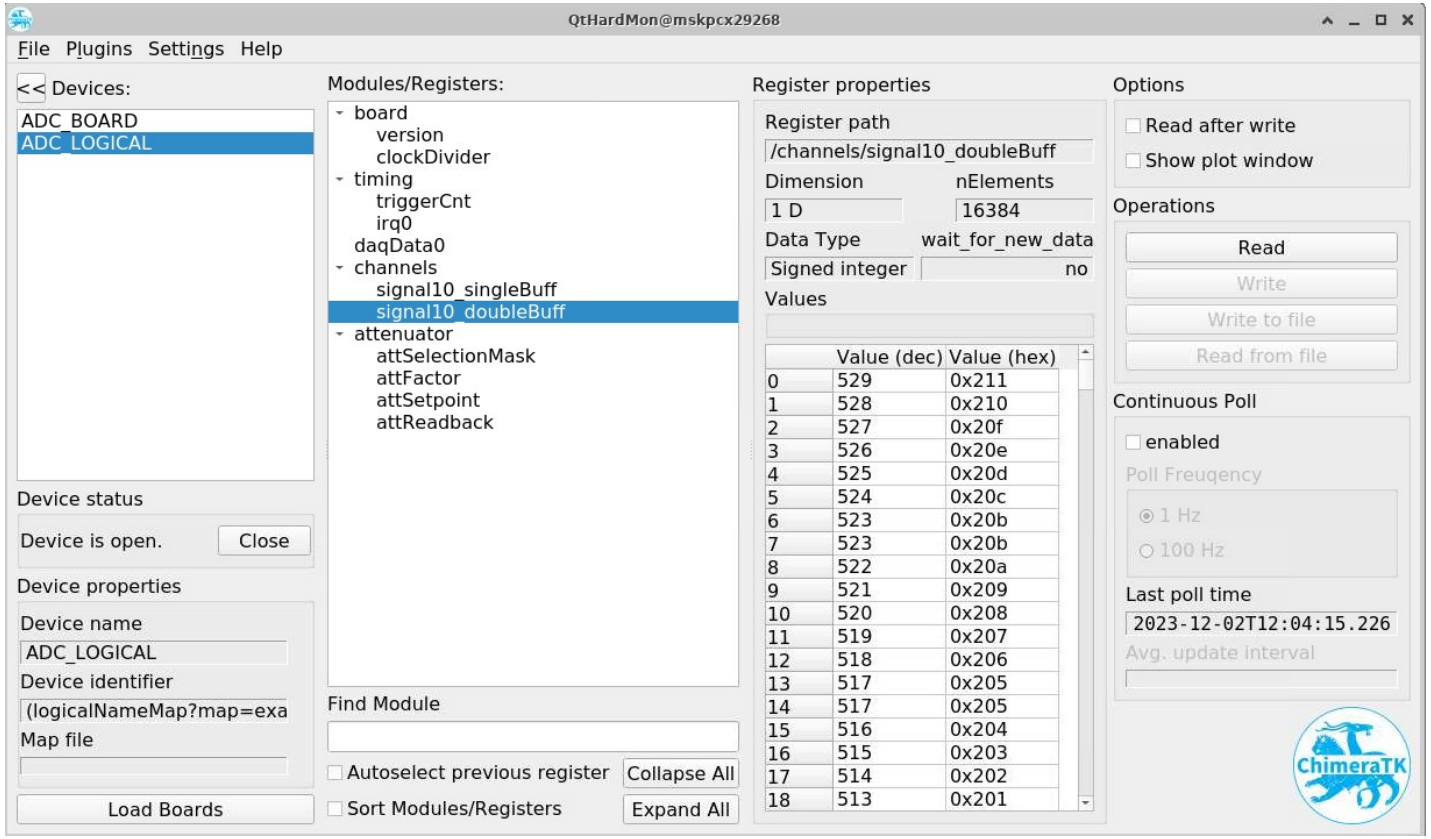
```
# devices.dmap
ADC_BOARD (xdma:xdma/slot3?map=example_dwc8vm1.mapp)
```

```
@MAPFILE_REVISION 2.2.0-2-g3cada082
# module.name           numElements  address   size  BAR  bits frac sign  mode
BSP.ID                            1  0x00000000   4    0   32    0    0   RO
ch0_top.BSP                   19201  0x00000000 76804  0   32    0    0   RW
BSP.VERSION                       1  0x00000004   4    0   32    0    0   RO
BSP.PRJ_ID                        1  0x00000008   4    0   32    0    0   RO
BSP.PRJ_VERSION                   1  0x0000000C   4    0   32    0    0   RO
BSP.PRJ_SHASUM                    1  0x00000010   4    0   32    0    0   RO
BSP.PRJ_TIMESTAMP                 1  0x00000014   4    0   32    0    0   RO
BSP.SCRATCH                       1  0x00000018   4    0   32    0    0   RW
BSP.RESET_N                       1  0x0000001C   4    0    1    0    0   RW
BSP.CLK_MUX                       6  0x00000020  24    0    2    0    0   RW
BSP.CLK_SEL                       1  0x00000038   4    0    1    0    0   RW
BSP.CLK_RST                       1  0x0000003C   4    0    1    0    0   RW
BSP.CLK_FREQ                      4  0x00000040  16    0   32    0    0   RO
BSP.CLK_ERR                       1  0x00000050   4    0    1    0    0   RO
BSP.SPI_DIV_SEL                   1  0x00000054   4    0    2    0    0   RW
BSP.SPI_DIV_BUSY                  1  0x00000058   4    0    1    0    0   RO
BSP.ADC_ENA                       1  0x0000005C   4    0    1    0    0   RW
BSP.ADC_IDELAY_CNT                5  0x00000060  20    0    9    0    0   RO
BSP.ADC_REVERT_CLK                1  0x00000074   4    0    5    0    0   RW
BSP.SPI_ADC_SEL                   1  0x00000078   4    0    3    0    0   RW
BSP.SPI_ADC_BUSY                  1  0x0000007C   4    0    1    0    0   RO
BSP.ADC_DELAY                    10  0x00000080  40    0    8    0    0   RW
BSP.DAC_ENA                       1  0x000000A8   4    0    1    0    0   RW
BSP.DAC_IDELAY_INC                1  0x000000AC   4    0    1    0    0   RW
BSP.DAC_IDELAY_CNT                1  0x000000B0   4    0    9    0    0   RO
BSP.DDR_CALIB_DONE                1  0x000000B4   4    0    1    0    0   RO
BSP.BOOT_STATUS                   1  0x000000B8   4    0    1    0    0   RW
...
```

Logical Device

Logical name map - abstract technical firmware details

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- Extract channels
- Double buffering
- ...



Use with **same DeviceAccess API** as raw device

⟹ Use same tools for testing!

⟹ Enables **better abstraction** in server logic!

Logical Device

Logical name map (2)

- **Select & rename registers**
- **(Re-)define read/write access**
  server initialises all writeable automatically!

```
<logicalNameMap>
    <module name="board">
        <redirectedRegister name="version">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>BSP/PRJ_VERSION</targetRegister>
        </redirectedRegister>
        <redirectedRegister name="clockDivider">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>TIMING/DIVIDER_VALUE</targetRegister>
            <plugin name="forceReadOnly"/>
        </redirectedRegister>
    </module>

...
```

Logical Device

Logical name map (3)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- **Adjust data types**
- **Define new variables and constants**
- **Apply simple formulas**

```
...
    <module name="attenuator">
        <variable name="attFactor">
            <type>float32</type>
            <value>0</value>
        </variable>
        <redirectedRegister name="attSetpoint">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>RTM/ATT_VAL</targetRegister>
            <plugin name="math">
                <parameter name="enable_push_parameters"/>
                <parameter name="formula"> f*x </parameter>
                <parameter name="f">/attenuator/attFactor</parameter>
            </plugin>
        </redirectedRegister>
        <redirectedRegister name="attReadback">
            <targetDevice>ADC_BOARD</targetDevice>
            <targetRegister>RTM/ATT_VAL</targetRegister>
            <plugin name="forceReadOnly"/>
            <plugin name="typeHintModifier">
                <parameter name="type">int32</parameter>
            </plugin>
        </redirectedRegister>
    </module>
</logicalNameMap>
```

Logical Device

Logical name map (4)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- **Extract channels**

```
<logicalNameMap>
     ...
      <module name="channels">
       <redirectedChannel name="signal10_singleBuff">
           <targetDevice>ADC_BOARD</targetDevice>
           <targetRegister>DAQBUF/DAQ_CTRL_BUF0</targetRegister>
           <targetChannel>10</targetChannel>
           <plugin name="typeHintModifier">
               <parameter name="type">int32</parameter>
           </plugin>
       </redirectedChannel>
    </module>
</logicalNameMap>
```

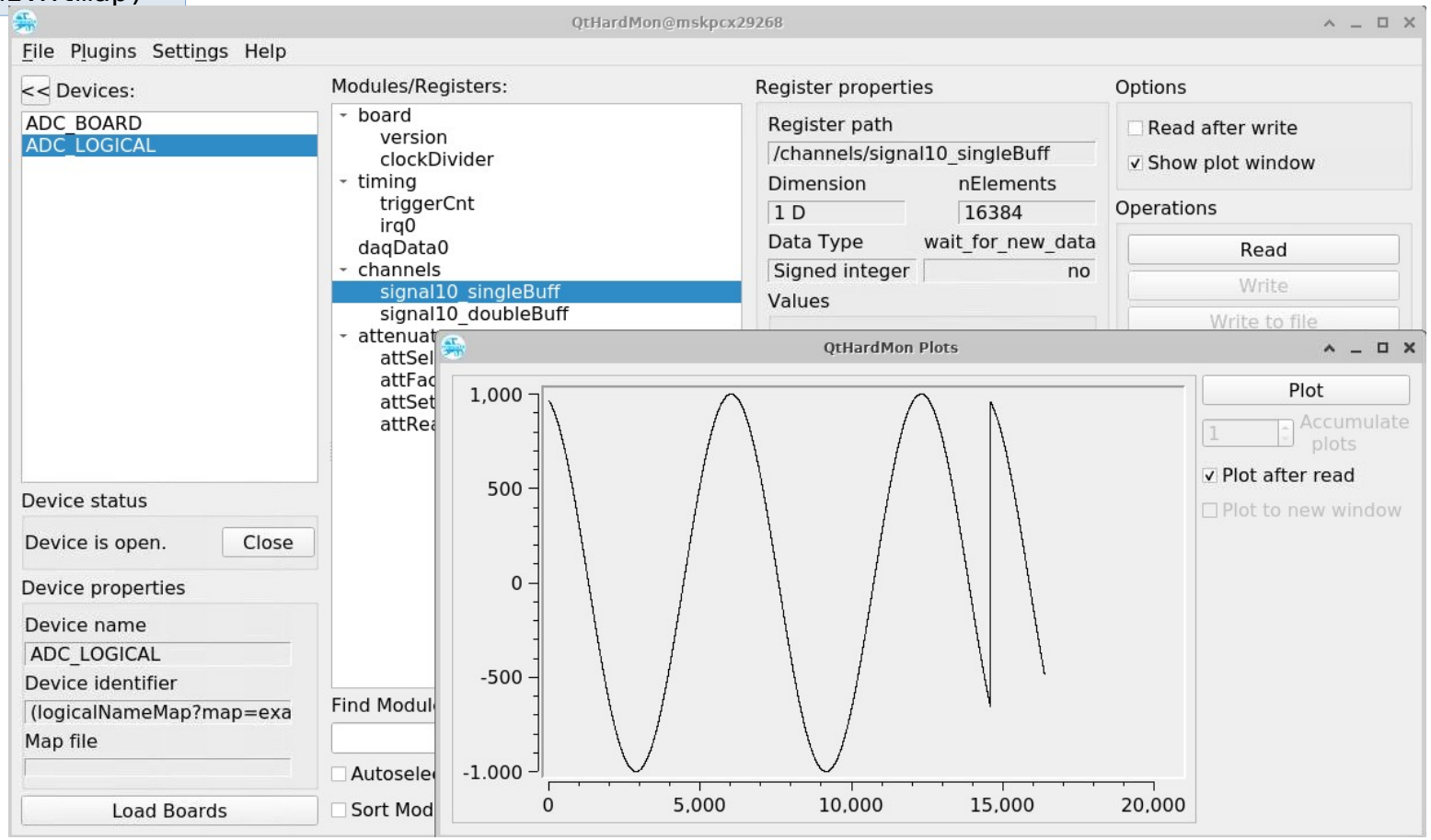
  - FPGA application output = 16 signals(time)
  - Memory bandwidth enforces that signals values of same time are next to each other $\Longrightarrow$ 2D matrix of multiplexed data

  DeviceAccess:
  - 2D RegisterAccessor for demultiplexed 2D matrix of data
  - **`'redirectedChannel'`** selects channel $\Longrightarrow$ 1D RegisterAccessor

Test double buffering – demonstration of problem

DeviceAccess

```
# devices.dmap
#ADC_BOARD (xdma:xdma/slot3?map=example_dwc8vm1.mapp)
ADC_BOARD (sharedMemoryDummy?map=example_dwc8vm1.mapp)
ADC_LOGICAL (logicalNameMap?map=example_dwc8vm1.xlmap)
```

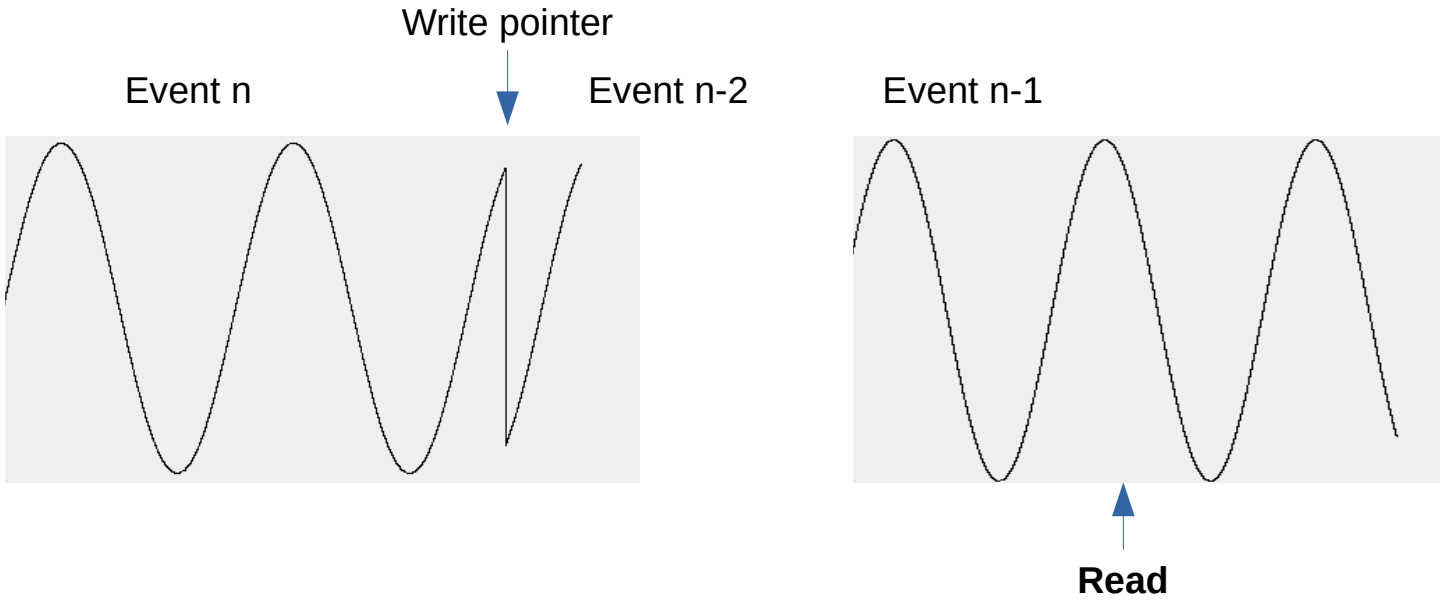


Motivation - double buffering

DeviceAccess

Issue: No complete event to read

$\Longrightarrow$ need two buffers and register with currently written buffer



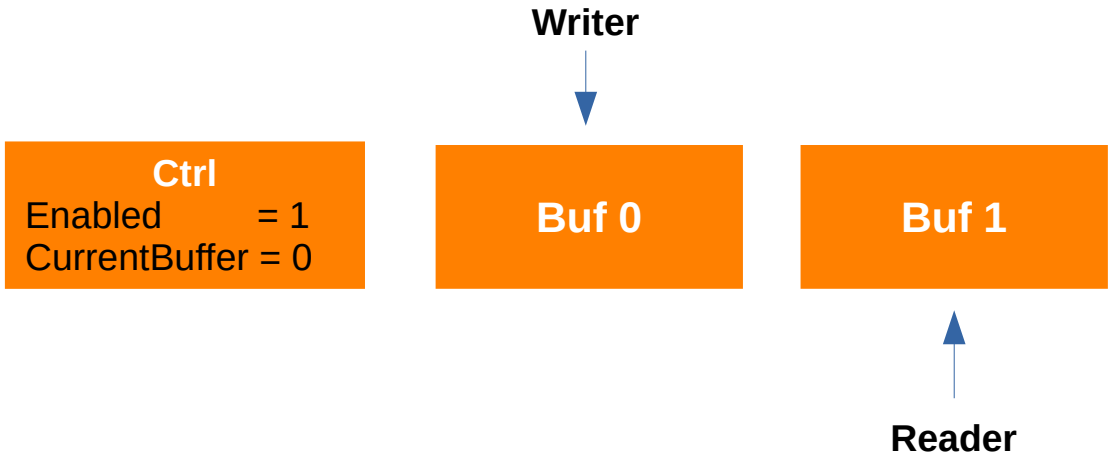
Issue: Reading server is not real-time, runs asynchronously to writing firmware

$\Longrightarrow$ need control register, to disable buffer swapping while reading

Motivation - double buffering

DeviceAccess

Reader and writer activity are not synchronized, but we want consistent data.
⟹ Need two buffers and two control registers!



Protocol for Reader (DeviceAccess)
- Disable buffer swapping (write Enabled register)
- Read current buffer number
- Read buffer contents of other buffer
- Enable buffer swapping

Protocol for Writer (Firmware)
- Write to current buffer
- Read Enabled register
- If buffer swapping allowed: toggle buffer number

Logical Device

Logical name map (5)

- Select & rename registers
- (Re-)define read/write access
  server initialises all writeable automatically!
- Adjust data types
- Define new variables and constants
- Apply simple formulas
- Extract channels
- **Double buffering**
- Extract elements from arrays
- Mono-stable triggers
- Extract bits
- Extract and thread-safely read-modify-write bit ranges (new)
- Planned support for Xilinx ring buffer IP core

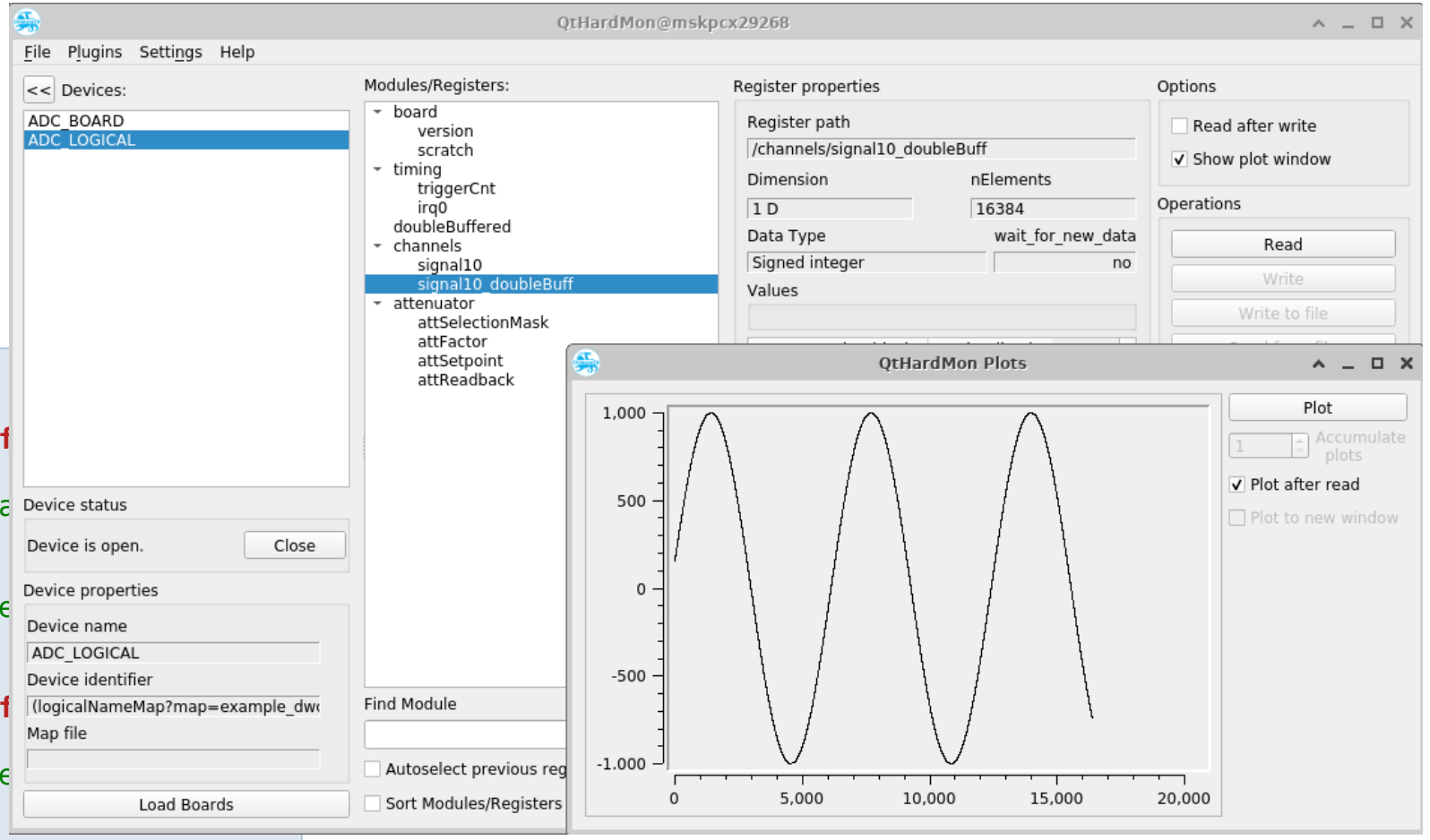
```
<logicalNameMap>
    ...

    <redirectedRegister name="daqData0">
        <targetDevice>ADC_BOARD</targetDevice>
        <targetRegister>/DAQBUF/DAQ_CTRL_BUF0</targetRegister>
        <plugin name="doubleBuffer">
            <parameter name="secondBuffer">/DAQBUF/DAQ_CTRL_BUF1</parameter>
            <parameter name="currentBufferNumber">/DAQ/ACTIVE_BUF</parameter>
            <parameter name="enableDoubleBuffering">
                /DAQ/DOUBLE_BUF_ENA</parameter>
            <parameter name="daqNumber">0</parameter>
        </plugin>
    </redirectedRegister>
</logicalNameMap>
```

Double buffering is completely hidden from server application!
Server uses only register ‘doubleBuffered’

Test double buffering – demonstration of solution

DeviceAccess

```
...
<module name="channels">
    <redirectedChannel name="signal10_singleBuff
        <targetDevice>ADC_BOARD</targetDevice>
        <targetRegister>DAQBUF/DAQ_CTRL_BUF0</ta
        <targetChannel>10</targetChannel>
        <plugin name="typeHintModifier">
            <parameter name="type">int32</parame
        </plugin>
    </redirectedChannel>
    <redirectedChannel name="signal10_doubleBuff
        <targetDevice>this</targetDevice>
        <targetRegister>/daqData0</targetRegiste
        <targetChannel>10</targetChannel>
        <plugin name="typeHintModifier">
            <parametername="type">int32</parameter>
        </plugin>
    </redirectedChannel>
</module>
```

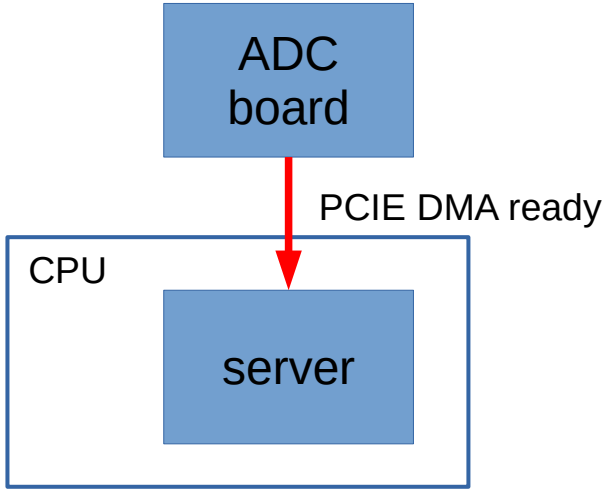



```
# devices.dmap
#ADC_BOARD (xdma:xdma/slot3?map=example_dwc8vm1.mapp)
ADC_BOARD (sharedMemoryDummy?map=example_dwc8vm1.mapp)
ADC_LOGICAL (logicalNameMap?map=example_dwc8vm1.xlmap)
```

User Interrupts with PCIE

Motivation

- machine synchronous, get rid of polling
- Simple configuration (no delay to be configured)
- Support complex non-periodic triggering requirements



DeviceAccess will provide events with Accessors in mode = wait_for_new_data

User Interrupts with PCIE

DeviceAccess

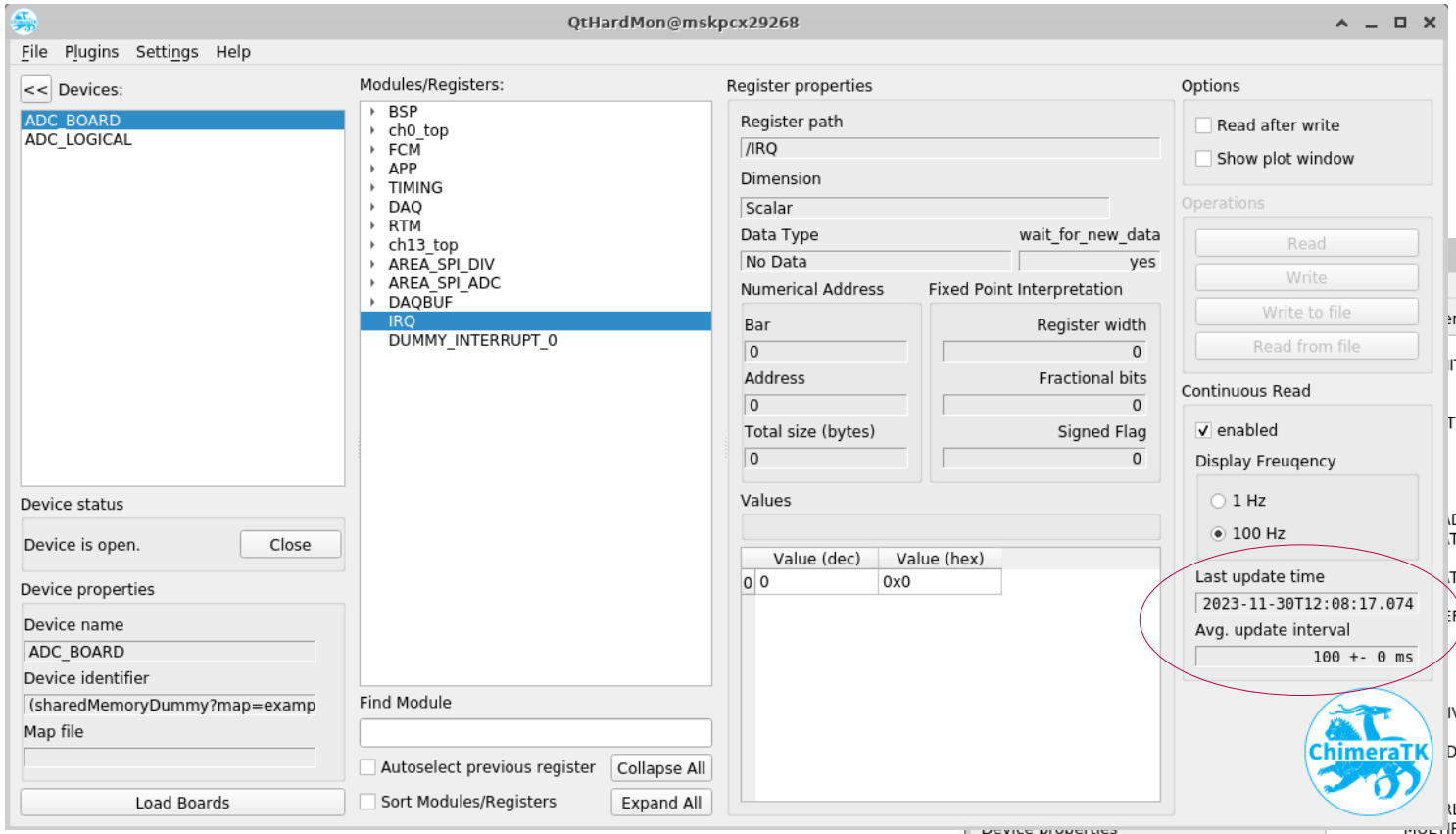
```
BSP.LLL_STATUS                  6    0x0000015C      24        0      32        0        0      RO
TIMING.TRIGGER_CNT_IRQ          4    0x0080403C      16        0      32        0        0      INTERRUPT0
...
IRQ                             0    0                0        0       0        0        0      INTERRUPT0
```

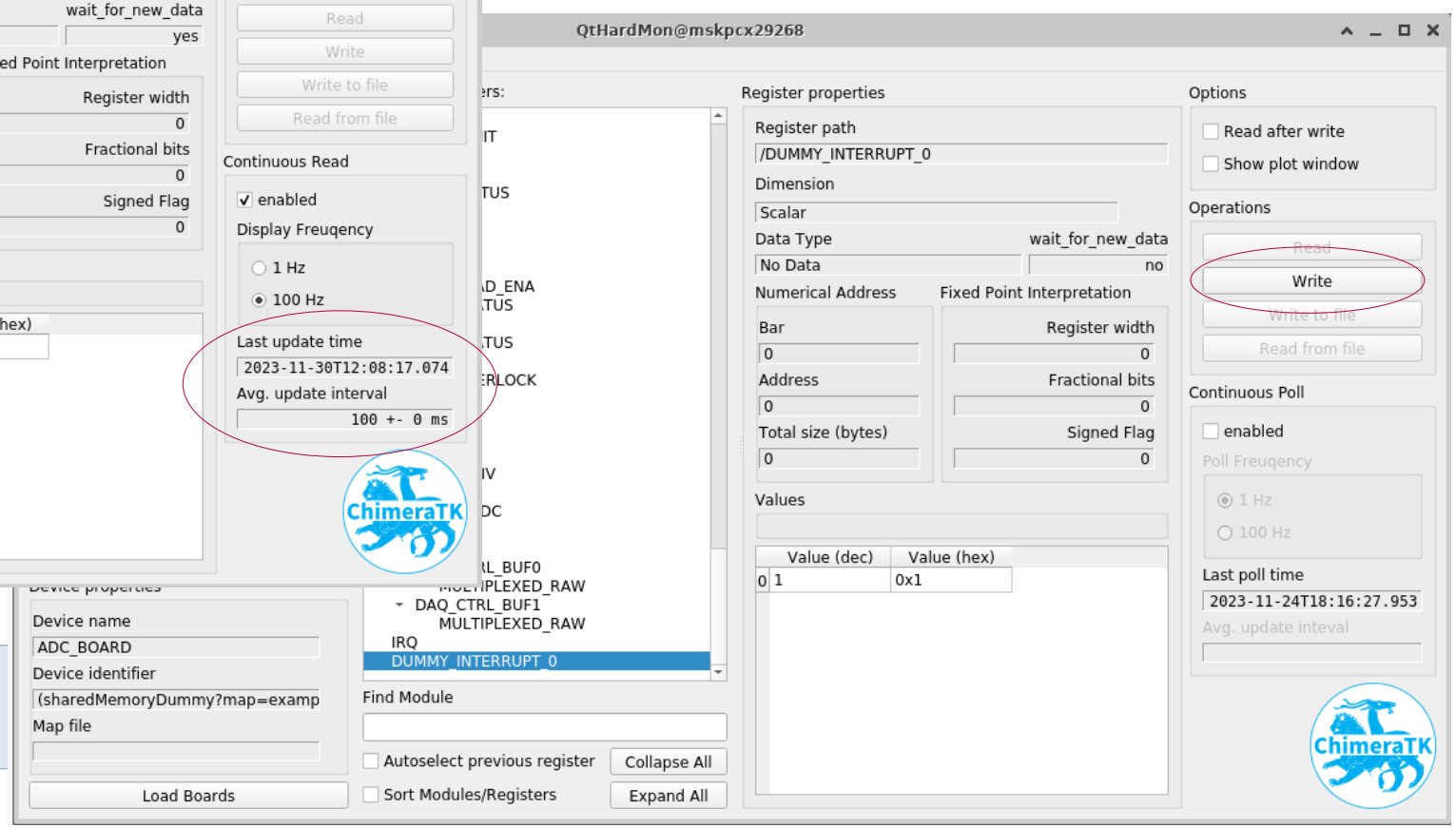
- Special syntax in Map-files for user interrupts
- Connect registers to interrupt (last column)
- Supported by xDMA backend
- DeviceAccess RegisterAccessors support async mode!

```python
#! /usr/bin/python3

import deviceaccess as da
import numpy as np

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

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


# read initial value: this is polled
triggerCounters.read()

while True :
        triggerCounters.read()
        print('Trigger, counter[2] = ' + str(triggerCounters[2]))
```

User Interrupts – Testing with SharedMemoryDummy

DeviceAccess





Two DeviceAcc backend instances on same SHM:

- QtHardMon & QtHardmon.
- QtHardMon & python script,
- QtHardMon & server,
- …

```
#devices.dmap
ADC_BOARD (sharedMemoryDummy?map=example_dwc8vm1.mapp)
ADC_LOGICAL (logicalNameMap?map=example_dwc8vm1.xlmap)
```

GenericDeviceServer

Configure GenericDeviceServer

- Prepare devices.dmap
- **Initialization script**
  Using DeviceAccess Python bindings on raw device
- **Add Logical Device: *.xlmap**
- **Configure readout trigger**
  - Simple PeriodicTimer in software
  - **Interrupt**
  - Another backend

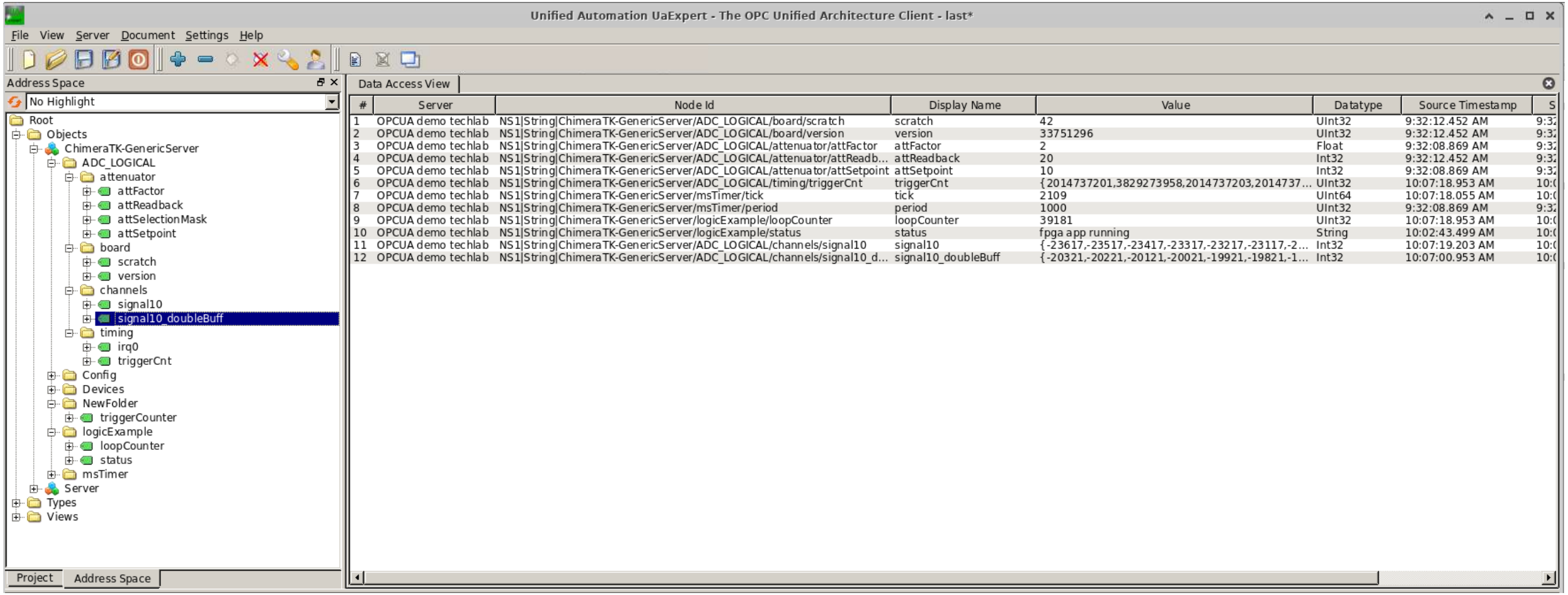
generic_chimeratk_server_configuration.xml

```
<configuration>
    <variable name="devices" type="string">
        <value i="0" v="ADC_LOGICAL"/>
    </variable>

    <module name="ADC_LOGICAL">
        <variable name="triggerPath" type="string"
                  value="/ADC_LOGICAL/timing/irq0" />
        <variable name="pathInDevice" type="string" value=""/>
        <variable name="initScript" type="string" value="./initscript.py"/>
    </module>
</configuration>
```

GenericDeviceServer with OPC UA Adapter

Run it

- Fully working device server just by configuration
- **GenericDeviceServer in use at FLASH master oscillator!**



OPC UA Control System Adapter

Configuration (optional)

Goal: Integrate control application into existing control system

- Adapt server instance name!
  We might have several instances running even in different facilities
- Port & security settings
- Organize and rename process variables according to requested naming convention

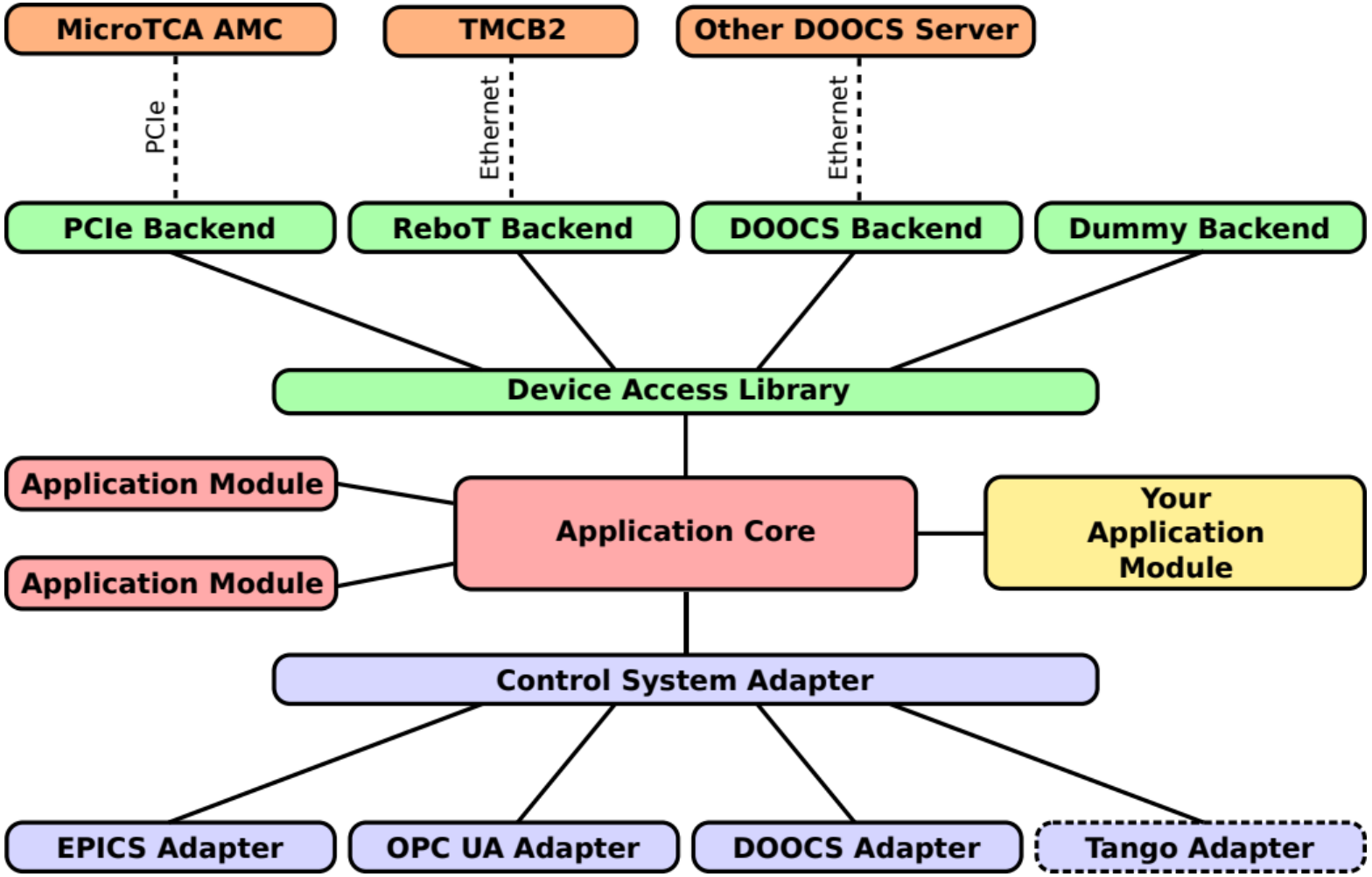
generic_chimeratk_server_mapping.xml

```
<csa:uamapping xmlns:csa="https://github.com/ChimeraTK/ControlSystemAdapter-OPC-UA-Adapter">
    <config>
        <server applicationName="ADC_DemoServer" port="4840"/>
        <security certificate="opcua_certs/server_cert.der" privatekey="opcua_certs/server_key.der"
          trustlist="opcua_certs/trust" blocklist="opcua_certs/blocklists" issuerlist="opcua_certs/issuer" />
        <login username="user" password="test"/>
    </config>
    <process_variable sourceName="ADC_LOGICAL/timing/triggerCnt" copy="true">
        <name>triggerCounter</name>
        <destination>NewFolder</destination>
    </process_variable>
</csa:uamapping>
```

ApplicationCore

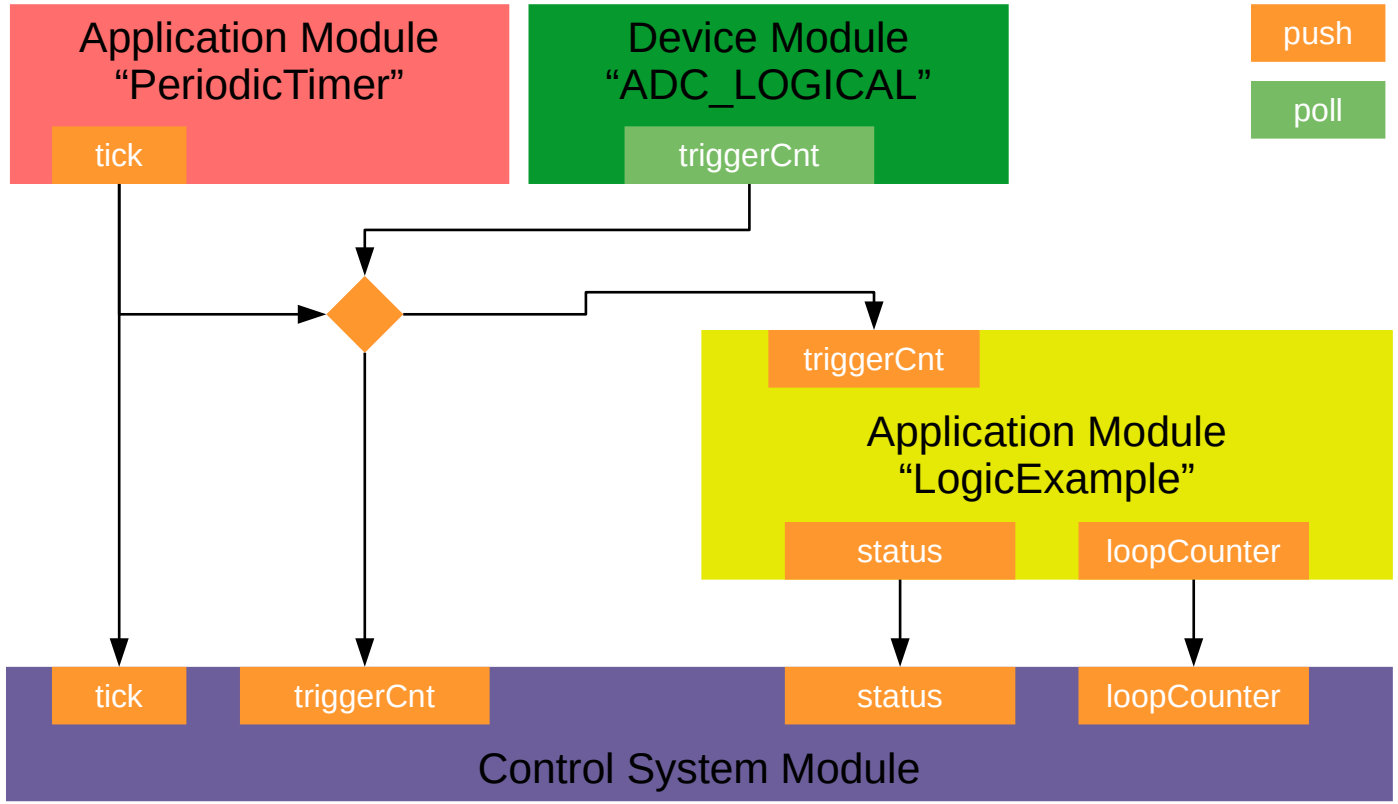
Extend server with business logic

ChimeraTK overview reminder



Extend server with business logic

ApplicationModule in C++



Register = Process Variable = CS Property

- New business logic as ApplicationModule
- Self-contained
- Runs in its own thread
- Modern inter-thread communication with lock-free queues
- Connection code is automatically generated
- Variables of same path/name are automatically connected
- Push-inputs are automatically connected to the trigger and the polled register

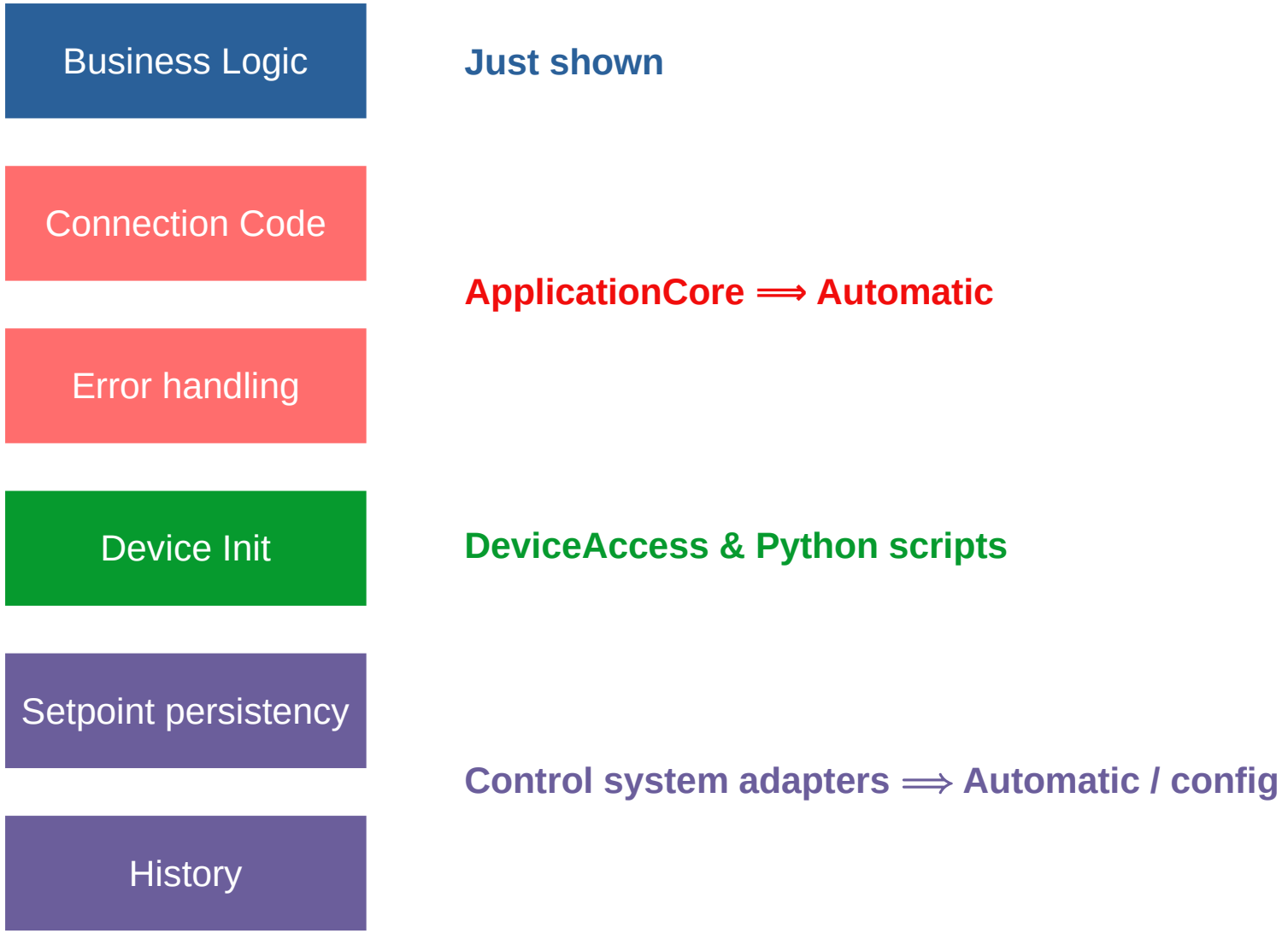
Extend server by business logic

ApplicationCore

```
struct LogicExample : public ChimeraTK::ApplicationModule {
    using ctk::ApplicationModule::ApplicationModule;
    ChimeraTK::ArrayPushInput<uint32_t> timingCnt{this, "/ADC_LOGICAL/timing/triggerCnt", "", 4, "hw trigger counters"};
    ChimeraTK::ScalarOutput<std::string> status{this, "status", "(no unit)", "fpga app status"};
    ChimeraTK::ScalarOutput<uint32_t> loopCounter{this, "loopCounter", "", ""};

    void mainLoop() override {
        // device is already initialized and current input values loaded
        uint32_t lastCnt = 0;
        while(true) {
            status = (timingCnt[1] != lastCnt) ? "fpga app running" : "fpga app not running";
            lastCnt = timingCnt[1];
            loopCounter++;
            writeAll();
            readAll();
        }
    }
};

struct ExtendedGenericApp : public ctk::Application {
    ...
    LogicExample sc{this, "logicExample", ""};
};
```

Extend server by business logic

What functionality do we need for a full Control System Application?



Control System Adapter for DOOCS

Configuration

doocs-generic-chimeratk-server01.conf
= general configuration as required by DOOCS

- Access control
- Port numbers
- Global history & autosave parameters

```
eq_conf:

oper_uid: -1
oper_gid: 422
xpert_uid:    0
xpert_gid:    0

eq_fct_name:    "NODENAME._SVR"
eq_fct_type:    1
{
SVR.STORE.RATE: 10
SVR.STORE.AUTO: 4
SVR.RPC_NUMBER: 610498009
SVR.FACILITY:   "TEST.DOOCS"
}
```

generic_chimeratk_server-DoocsVariableConfig.xml
= DoocsAdapter specific configuration

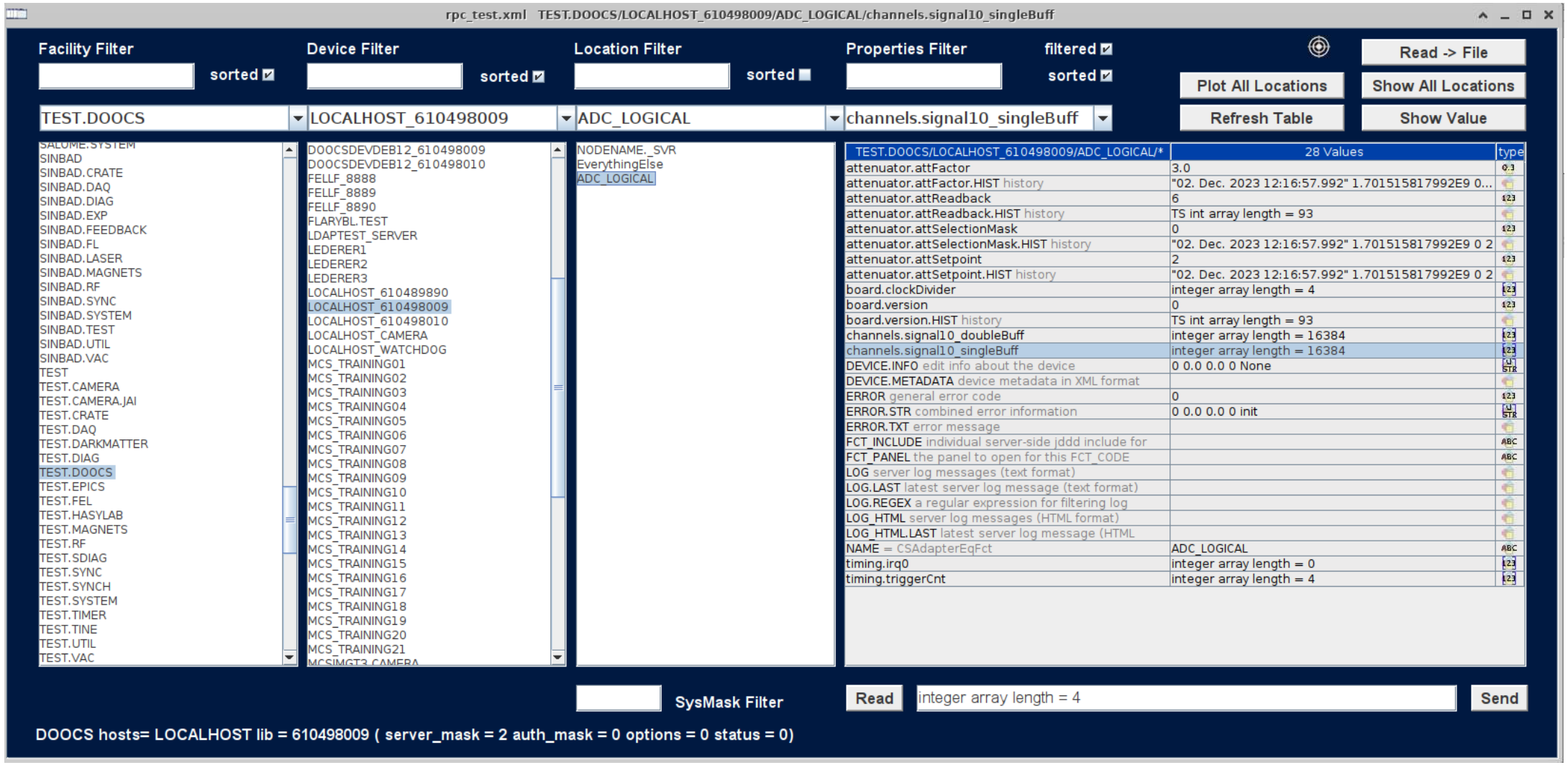
- Select, rename, reorganize registers into DOOCS properties
- History configuration (per property)
- ...

```
<?xml version="1.0" encoding="UTF-8"?>
<device_server
xmlns="https://github.com/ChimeraTK/ControlSystemAdapter-
DoocsAdapter">
    <location name="ADC_LOGICAL">
        <import>/ADC_LOGICAL</import>
    </location>
    <location name="EverythingElse">
        <import>/</import>
    </location>
</device_server>
```

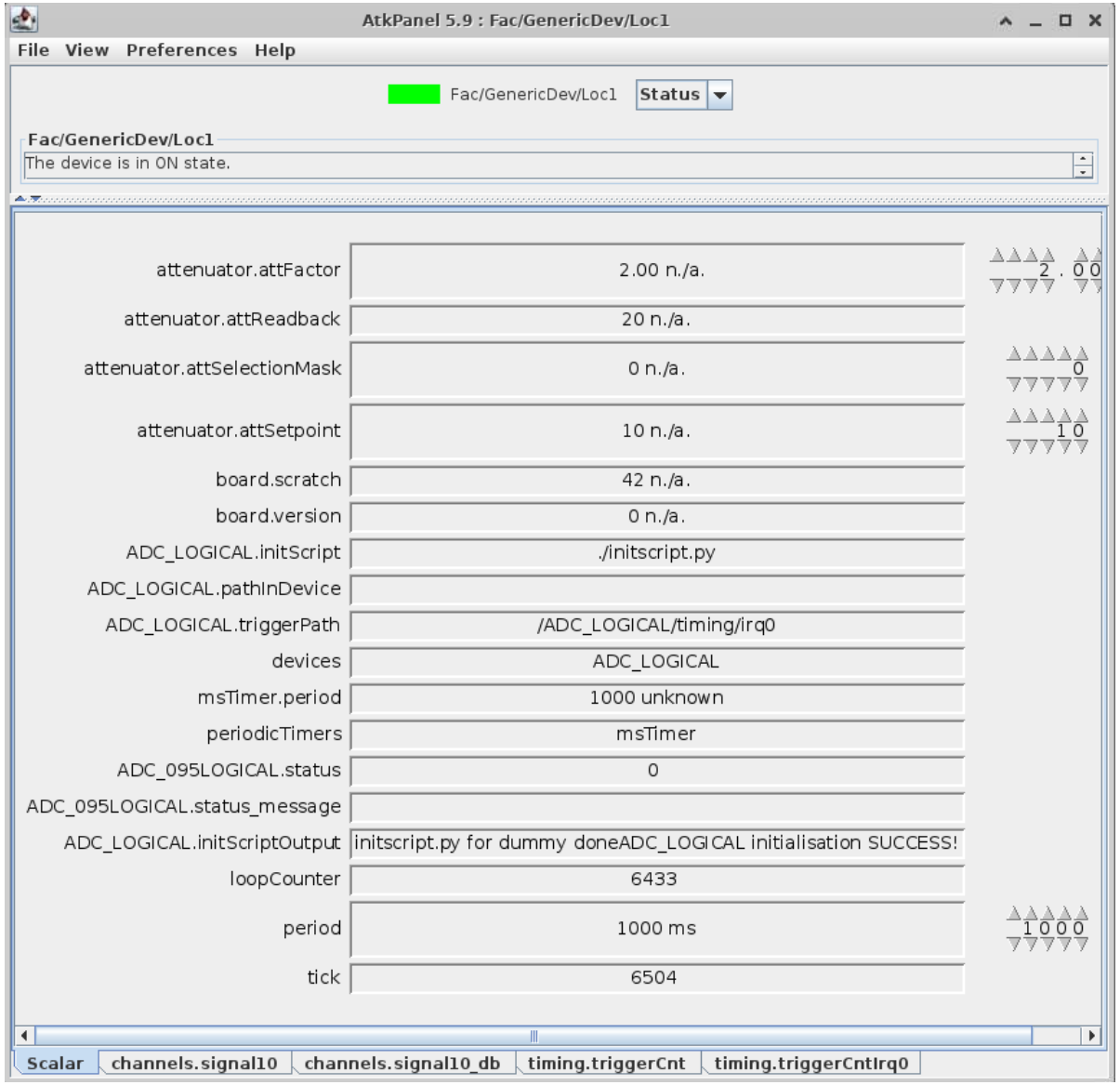
Control System Adapter for DOOCS

Build

```
$ cmake -DADAPTER="DOOCS" -S ../GenericDeviceServer/ . ; make
$ ./doocs-generic-chimeratk-server01
```



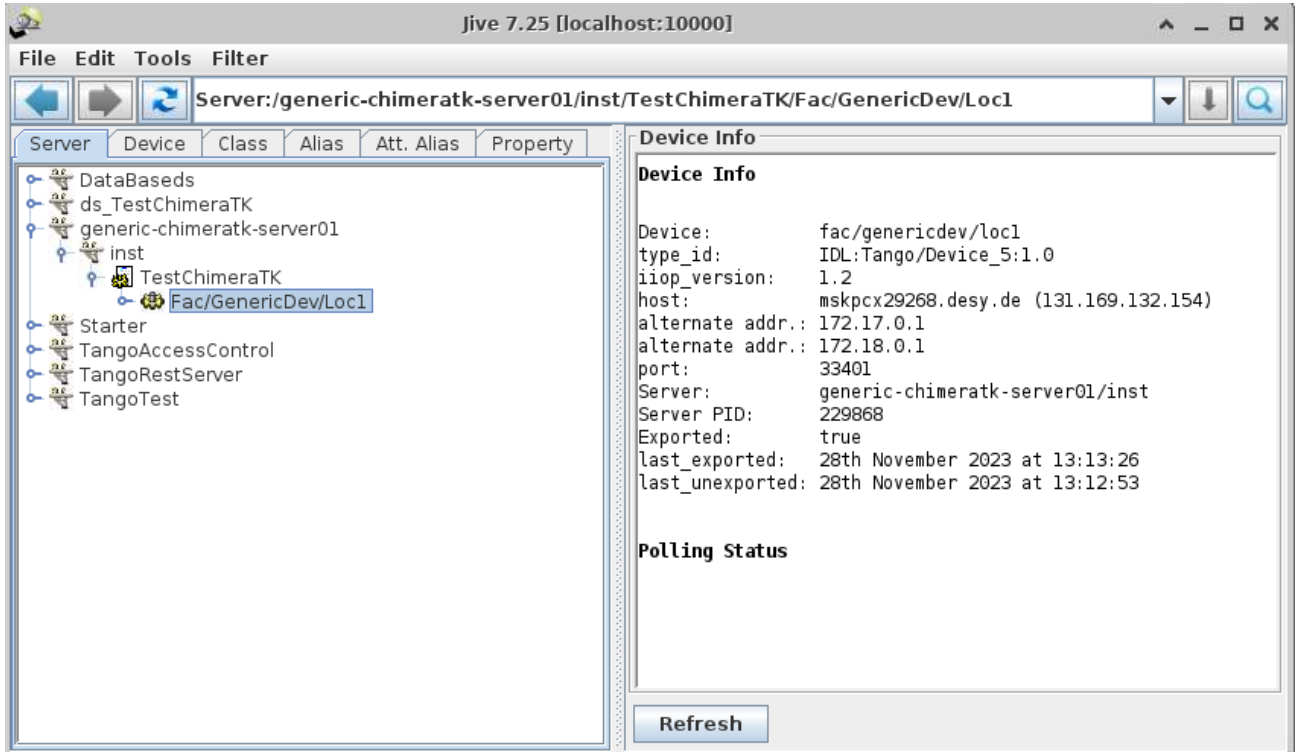
Control System Adapter for Tango



Currently under development by SOLEIL SYNCHROTRON

EXPERIMENTAL!

```
$ cmake -DADAPTER="TANGO" -S ../GenericDeviceServer/ . ; make
$ ./tango-generic-chimeratk-server01
```



Control System Adapter for EPICS

```
$ cmake -DADAPTER="EPICSIOC" -S ../GenericDeviceServer/ .
$ make
$ ./epics-generic-chimeratk-server01
```

```
< start.ioc
DoocsBackend::BackendRegisterer: registering backend type doocs
epics> < start.ioc
dbLoadDatabase("dbd/ChimeraTK.dbd",0,0)
ChimeraTK_registerRecordDeviceDriver(pdbbase)
chimeraTKConfigureApplication("ChimeraTKApp", 100)
# dbLoadRecords("db/sel-board.db","Server=VCT1,APP=ChimeraTKApp")
dbLoadRecords("db/sel-app.db","Server=VCT1,APP=ChimeraTKApp")
dbLoadRecords("db/sel-channel.db","Server=VCT1,ChName=Probe,ChNumber=0,APP=ChimeraTKApp")
iocInit()
Starting iocInit
############################################################################
## EPICS R3.16.2
## EPICS Base built May 23 2022
############################################################################
Warning: Duplicate EPICS CA Address list entry "192.168.115.255:5065" discarded
iocRun: All initialization complete
epics> All application modules are running.

epics> dbgrep VCT1/Probe*
VCT1/Probe/DAQ_Phase
VCT1/Probe/DAQ_Amplitude
VCT1/Probe/TableCosine
VCT1/Probe/TableSine
VCT1/Probe/FilterCoefficients
VCT1/Probe/DecimationFactor
VCT1/Probe/AmplitudeLimit
VCT1/Probe/AmplitudePrelimit
VCT1/Probe/AmplitudeTriggerLimit
VCT1/Probe/MonitorAmplitude
VCT1/Probe/MonitorPhase
VCT1/Probe/DCM_Enable
VCT1/Probe/VSshiftEnable
VCT1/Probe/AmplitudeLimitDisable
VCT1/Probe/AmplitudeTriggerLimitDisable
VCT1/Probe/DecimationMode
VCT1/Probe/AmplitudeLimitReached
VCT1/Probe/AmplitudePrelimitReached
VCT1/Probe/AmplitudeTriggerLimitReached
epics>
(llrfuser) mskcpukekvct | DeviceAccess-PythonBindings : bash | DebianPackagingScripts : bash
```

```
File  Edit  View  Bookmarks  Settings  Help
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableSine.VAL
3 0.866 -0.866 0
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableCosine.VAL
3 -0.5 -0.5 1
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caput -a VCT1/Probe/TableSine.VAL 3 0 0.866 -0.866
Old : VCT1/Probe/TableSine.VAL 3 0.866 -0.866 0
New : VCT1/Probe/TableSine.VAL 3 0 0.866 -0.866
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caput -a VCT1/Probe/TableCosine.VAL 3 1 -0.5 -0.5
Old : VCT1/Probe/TableCosine.VAL 3 -0.5 -0.5 1
New : VCT1/Probe/TableCosine.VAL 3 1 -0.5 -0.5
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableSine.VAL
3 0 0.866 -0.866
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$ caget -t VCT1/Probe/TableCosine.VAL
3 1 -0.5 -0.5
llrfuser@stfvtcpu:/var/epics-servers/VCT-SEL$
```

What’s new & Outlook

What's new

ApplicationCore 3: Fully automatic connection of variables

- PCIe user interrupts
- Double buffering plugin
- Bit range plugin
- Linux UIO backend (SoCs)
- Yocto Linux

Yocto + OPC UA GenericServer on SoC used at SOLEIL !

What's coming

- Interrupt controller handler: distributes selected logical interrupts
- LMap plugin for Xilinx ring buffer

Route to quickly get started

Many commonly used features already available in the framework

- Use what is there, don't re-invent the wheel
- Extensions and bug fixes welcome!

Solve implementation details on the low levels. Abstract them towards the higher levels.
Example: Server just reads double buffered register, handshake is hidden in DeviceAccess plugin

Don't take shortcuts when it comes to error handling

- "I can add that later" $\Longrightarrow$ might need a re-design
- DeviceAccess backends and ApplicationCore have a runtime error handling scheme

Summary

DeviceAccess: Flexible, configurable hardware access

- Native C++ with Python, Matlab and command line interface
- Extensible backend mechanism
- Graphical user interface QtHardMon
- Logical name mapping with powerful plugins

ApplicationCore

- Modular, multi-threaded, C++ control applications
- Automatic device error handling and data validity propagation

ControlSystemAdapter

- Native integration into DOOCS, EPICS, OPC UA or TANGO
- Configuration to adapt applications into a facility

Thank you

References

Source repositories
- ChimeraTK https://github.com/ChimeraTK
- GenericDeviceServer https://github.com/ChimeraTK/GenericDeviceServer.git

Ubuntu 20.04 packages DESY DOOCS repository

ChimeraTK Yocto Layer https://github.com/ChimeraTK/meta-chimeratk

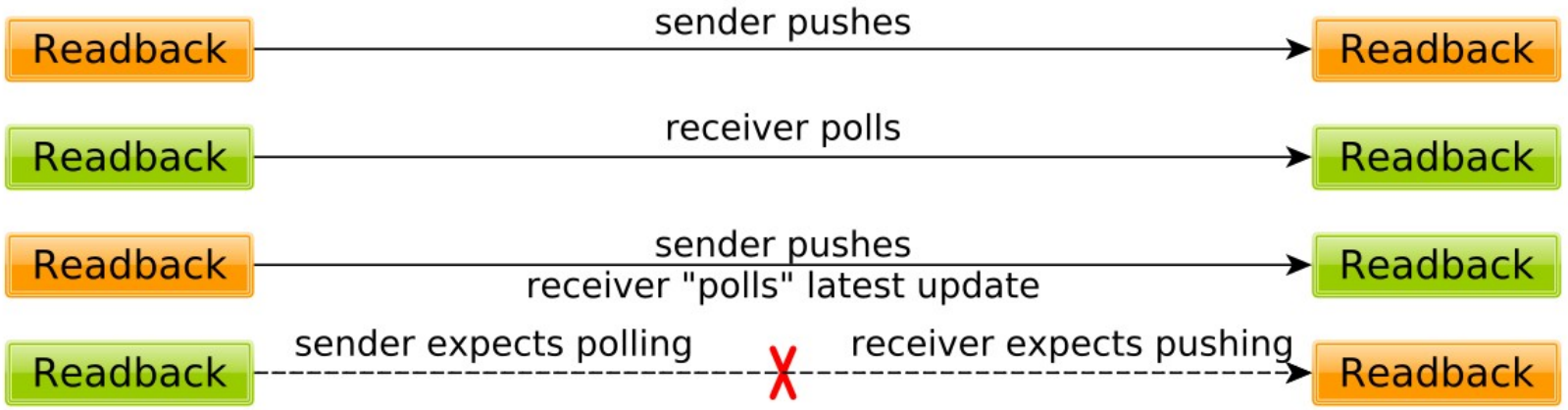
API documentation https://chimeratk.github.io/

Build details

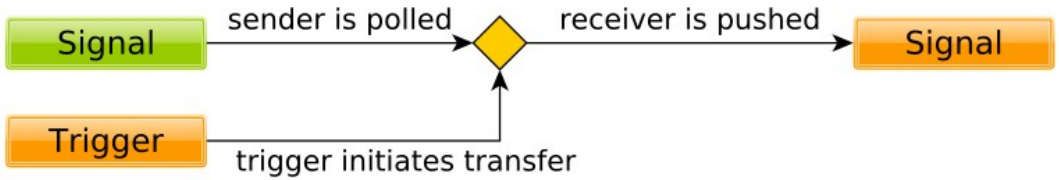
```
git clone https://github.com/ChimeraTK/GenericDeviceServer.git
git checkout drothe/mtcaws23-demo

# add DOOCS repository and key
wget -O - https://doocs-web.desy.de/pub/doocs/DOOCS-key.gpg.asc | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/doocs-keyring.gpg
sudo sh -c 'echo "deb https://doocs-web.desy.de/pub/doocs  $(lsb_release -cs) main" > /etc/apt/sources.list.d/doocs.list'

sudo apt install libchimeratk-applicationcore-dev libchimeratk-controlsystemadapter-opcuaadapter-dev libchimeratk-deviceaccess-doocsbackend-dev python3-mtca4upy
cmake   -DADAPTER="OPCUA"  -S ../GenericDeviceServer/ .
make
./opcua-generic-chimeratk-server01
```

Backup: Update modes

- Analogue to the client/server concept:
  - Client/server is about who initiates a connection
  - Update mode is about who initiates a data transfer



- A trigger makes it possible:



Backup: ChimeraTK Device Descriptor

Example device map file

```
#alias_name    device_descriptor
ADC_SLOT1      (pci:pcieunis1?map=my_adc_firmware.map)
ADC_SLOT2      (pci:pcieunis2?map=my_adc_firmware.map)
CONTROLLER     (pci:pcieunis3?map=my_controller_firmware.map)

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

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.