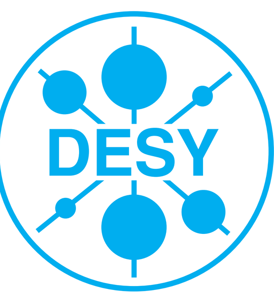
ApplicationCore: A Framework for Modern Control Applications at the Example of a Facility Independent LLRF Server

M. Hierholzer, M. Killenberg, C. Schmidt, N. Shehzad, T. Kozak, G. Varghese, M. Viti (DESY, Germany), M. Kuntzsch, R. Steinbrück (HZDR, Germany) S. Marsching (aquenos GmbH, Germany), A. Piotrowski (FastLogic, Poland), C. Iatrou, J. Rahm (TU Dresden, Germany)

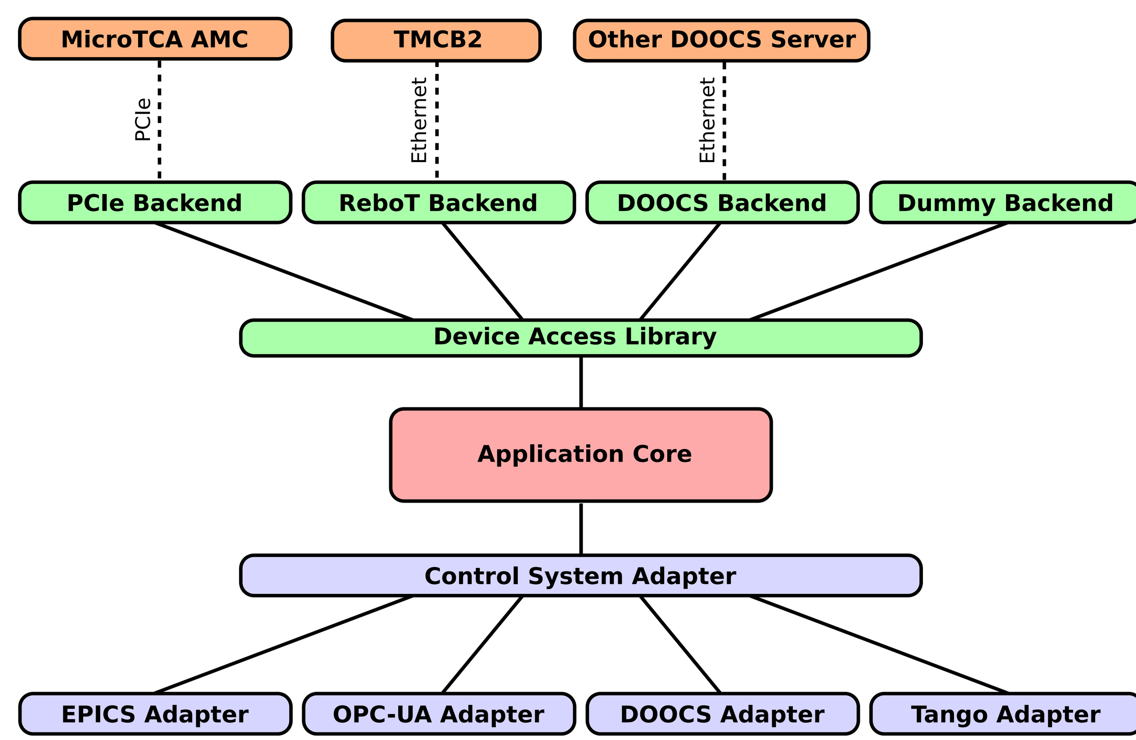


Presenter
Martin Hierholzer
martin.hierholzer@desy.de

Motivation

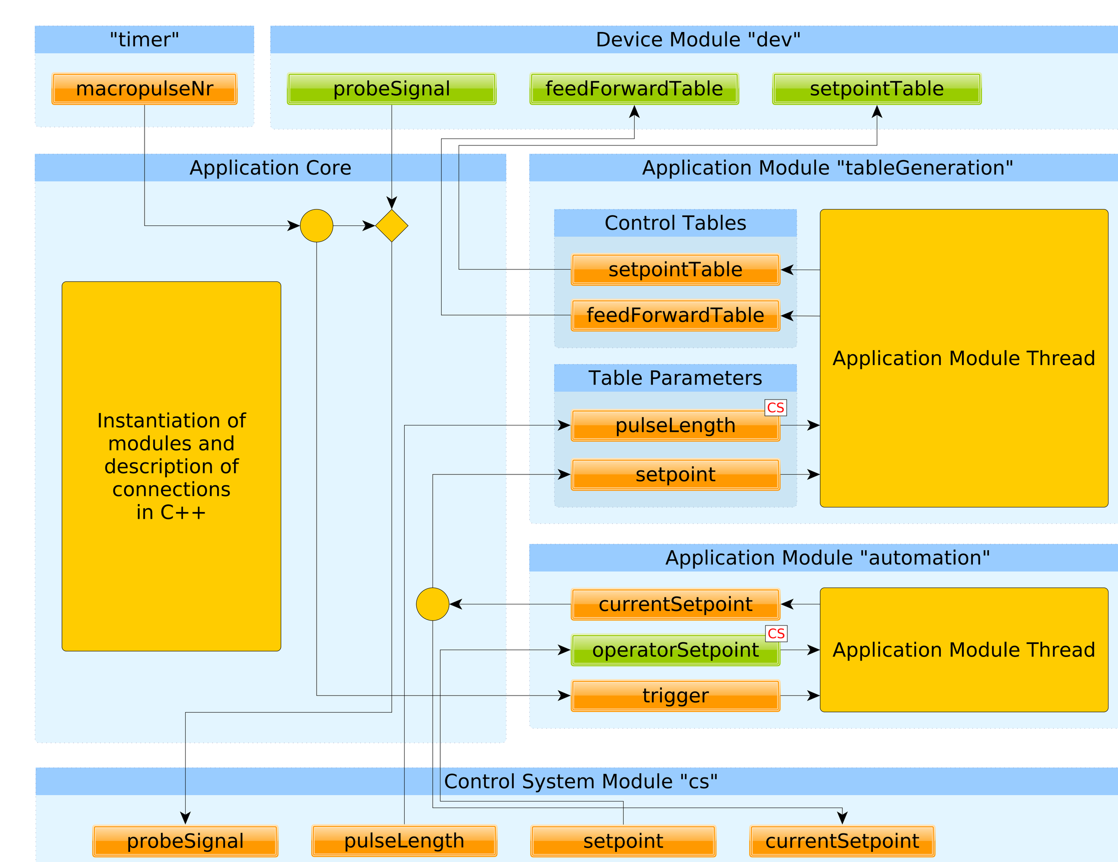
- MicroTCA-based LLRF system developed by DESY used in many facilities: FLASH, EuropeanXFEL and REGAE at DESY, ELBE at HZDR, FLUTE at KIT, TARLA at Ankara University (and more to come!)
- Facilities use different control system middleware: DOOCS at DESY, Siemens Simatic/WinCC with OPC UA at ELBE, EPICS 3 at FLUTE, EPICS 4 at TARLA
- RF control loop running in FPGA requires relative complicated software interface providing many control tables etc.
- ~ 700 control system variables for the single cavity controller, ~ 3000 variables for the vector sum controller
- Contains some complex algorithms (e.g. adaptive feed forward)
- Porting to different control systems by branching would be difficult to maintain
- Adaptation for different DOOCS-based systems at DESY already creates many problems: improve on modularity!

ChimeraTK abstraction layers



- DeviceAccess allows access to devices with different transport protocols
- ControlSystemAdapter allows to write applications for different middlewares
- ApplicationCore unifies both, and adds modularity and multi-threading

Concept of ApplicationCore



- Application consists of many modules: application modules with used code (algorithms), device modules and control system modules
- Main application part instantiates modules and defines connections

Code examples

```
struct Automation : public ApplicationModule {
  using ApplicationModule::ApplicationModule;
  ScalarPollInput<double> opSP{this, "opSP", "MV", "...", {"CS"}};
  ScalarOutput<double> curSP{this, "curSP", "MV", "..."};
  ScalarPushInput<int> trigger{this, "trigger", "", "..."};

  void mainLoop() {
    while(true) {
      trigger.read();      // block until next macro pulse
      opSP.readLatest();   // get current setpoint from panel
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();     // a waiting read() in "tableGeneration"
      }                    // will now return
    }
  }
};
```

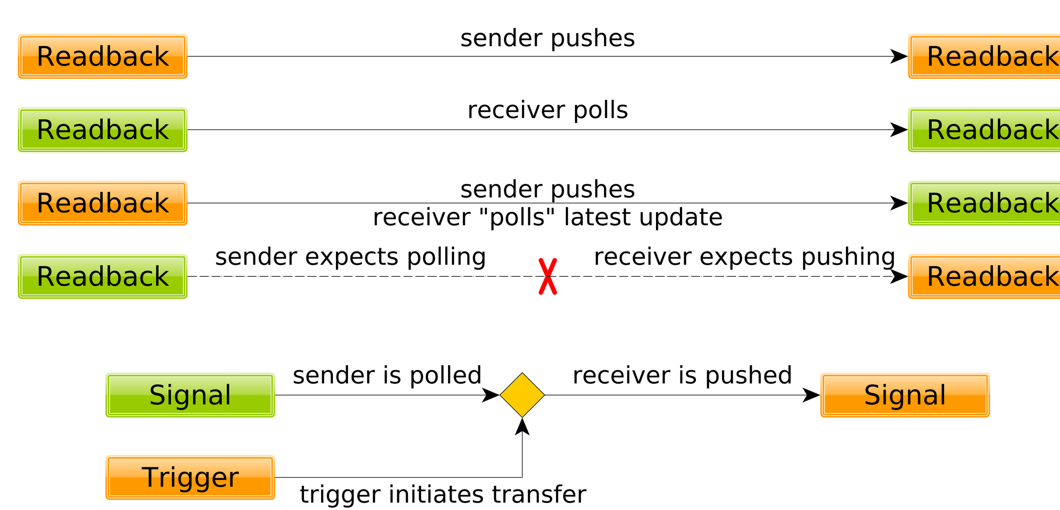
operatorSetpoint (CS) · currentSetpoint · trigger · Application Module Thread

```
// individual connection with distribution to two receivers
automation.curSP >> tableGeneration.tableParams.setpoint
                 >> cs("currentSetpoint");

// connect all control tables with device
tableGeneration.controlTables.connectTo(dev);

// connect all variables tagged with "CS" with control system
findTag("CS").connectTo(cs);

// special treatment for update mode
auto macropulseNr = timer("macropulseNr", typeid(int),1, ctk::UpdateMode::push);
macropulseNr >> automation.trigger;

// special treatment for direct device to control system connection
dev("probeSignal", typeid(int), 16384) [ macropulseNr ] >> cs("probeSignal");
```

Application modules

- Application modules have input and output "accessors"
- Accessors are same as in DeviceAccess:
  - can be used like normal variables
  - can be scalar or arrays of any fundamental data type
  - functions like read()/write() trigger data transfer
- Each application module has a thread running user code

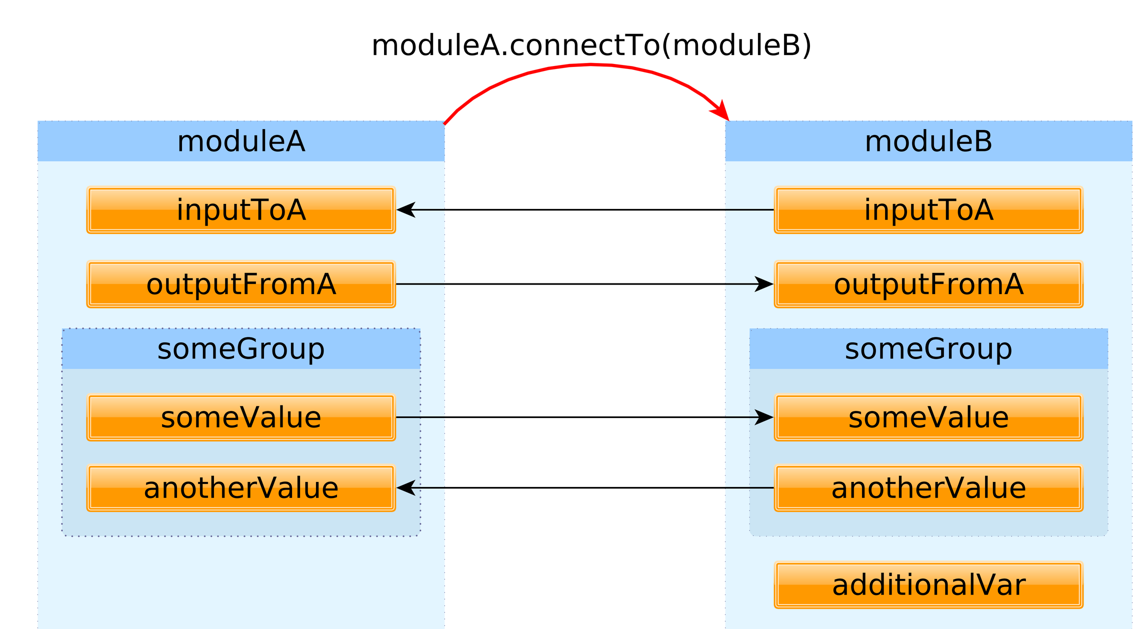
Update mode and trigger



- Each accessor must be defined either push or poll
- Triggers can be used to initiate data transfer e.g. from poll-only devices (not sending interrupts)
- Any push-type data source can be used as trigger
- LLRF server: use macropulse number from timing system as trigger for readback from FPGA

Variable groups

- Variables can be grouped for improved structure
- Group-wide transfers possible in owning thread: writeAll(), readAll(), readAny() etc.
- In description of connections, groups can be connected with a single command: connectTo()



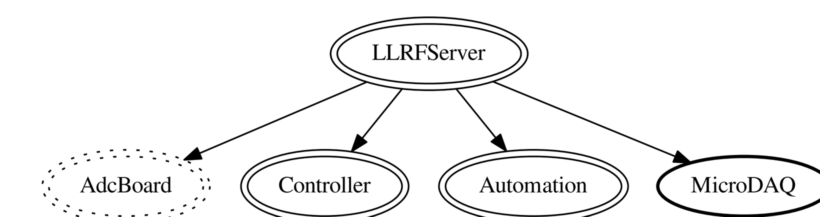
- Also module groups can be used with connectTo()

Multithreading

- Modern inter-thread communication and synchronisation based on queues and futures
- This is build into ApplicationCore, no need to deal with it in user code!
- Application modules just use their inputs and outputs, which will be connected automatically with queues
- Synchronisation works by means of blocking calls to read()

Modular structure of the LLRF server

Legend: ModuleGroup, ApplicationModule, VariableGroup, Hidden Hierarchy, Variable TAG1 TAG2



AdcBoard
attenuators · limiterActive · samplingScheme · Probe · Forward · Reflected · ...
Calibration · Filter · limiter · prelimiter · ChannelComputations
input · CascadeInput · output · Samples

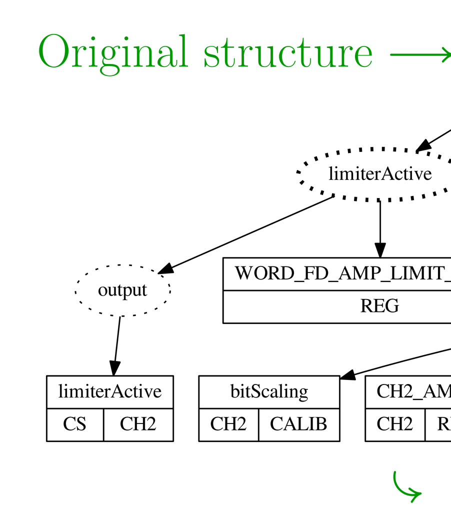
- 3 big module groups:
  - AdcBoard: direct signal monitoring
  - Controller: monitor feedback loop, generate control tables etc.
  - Automation: ramping (optional)
- Can have multiple AdcBoard groups for more channels (e.g. for mulit-cavity systems)
- MicroDAQ system writes data to HDF5 file

- AdcBoard has 8 (or 10) channels ("Probe", "Forward", ...), each with identical structure
- Additional board-wide modules like "limiterActive" also provide per-channel variables but access the same registers on hardware

Information model with tags

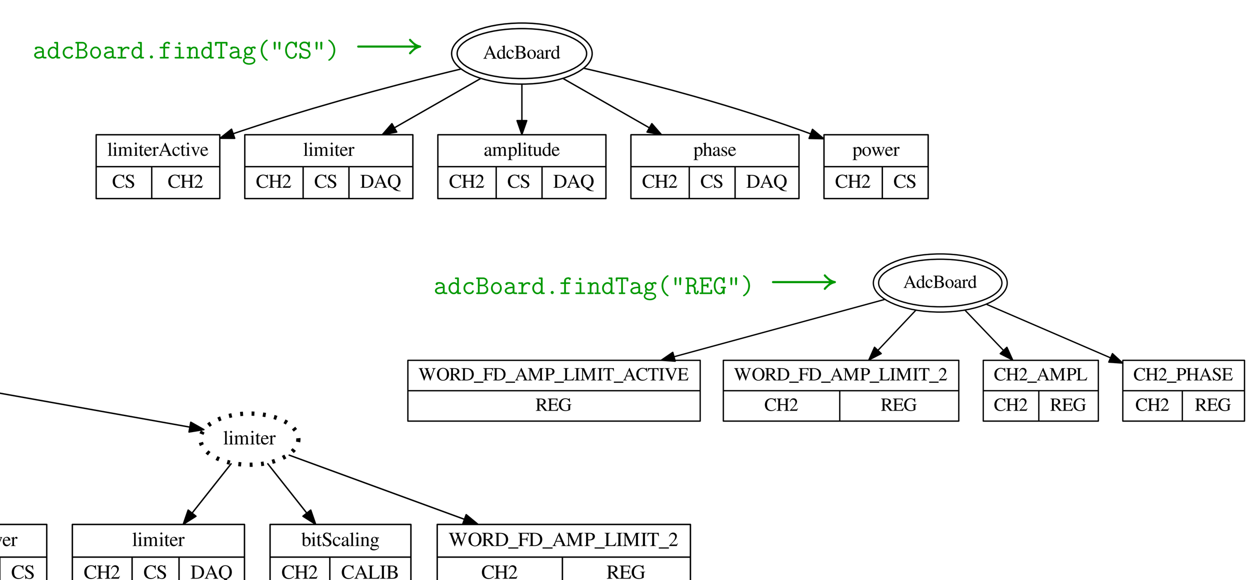
(model incomplete, full model too big to show)

Original structure →



limiterActive (CS, CH2) · bitScaling (CH2, CALIB) · CH2_AMPL (CH2, REG) · CH2_PHASE (CH2, REG) · amplitude (CH2, CS, DAQ) · phase (CH2, CS, DAQ) · power (CH2, CS) · limiter (CH2, CS, DAQ) · bitScaling (CH2, CALIB) · WORD_FD_AMP_LIMIT_2 (CH2, REG)

↳ tags ↲



Status of ApplicationCore and the LLRF server

- The ApplicationCore-based LLRF server has already been tested successfully in DOOCS environments
- Recently successful integration tests at HZDR/ELBE: driving a cavity with full control from WinCC over OPC UA
- CPU usage seems to be like the previous plain-DOOCS server (but runs faster due to improved multi-threading)
- ApplicationCore is available on Github: `http://github.com/ChimeraTK/ApplicationCore`
- Complete and working example available (see code on the left side)
- Documentation (work in progress): `http://chimeratk.github.io`

HELMHOLTZ ASSOCIATION · HZDR · FAST LOGIC · TECHNISCHE UNIVERSITÄT DRESDEN · aquenos software & more

LaTeX TikZposter