Introduction to ApplicationCore.

Martin Hierholzer

2017-09-21

DESY MSK

How it all began...

- Kick-off meeting for the OPC-UA adapter with TU-DD (Summer 2016)
- Why do we treat control system variables and registers differently?

How it all began...

- Kick-off meeting for the OPC-UA adapter with TU-DD (Summer 2016)
- Why do we treat control system variables and registers differently?
- Control system variable $\Rightarrow$ Server
- DeviceAccess register $\Rightarrow$ Client
- Conceptionally the same, can sometimes even be exchanged!

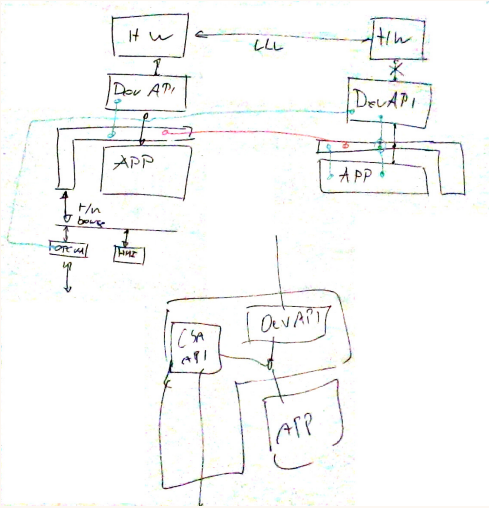
How it all began...

- Kick-off meeting for the OPC-UA adapter with TU-DD (Summer 2016)
- Why do we treat control system variables and registers differently?
- Control system variable $\Rightarrow$ Server
- DeviceAccess register $\Rightarrow$ Client
- Conceptionally the same, can sometimes even be exchanged!
- DOOCS example:
  - Client $\xrightarrow{\text{write via RPC}}$ Server
  - Server $\xrightarrow{\text{send via 0MQ}}$ Client

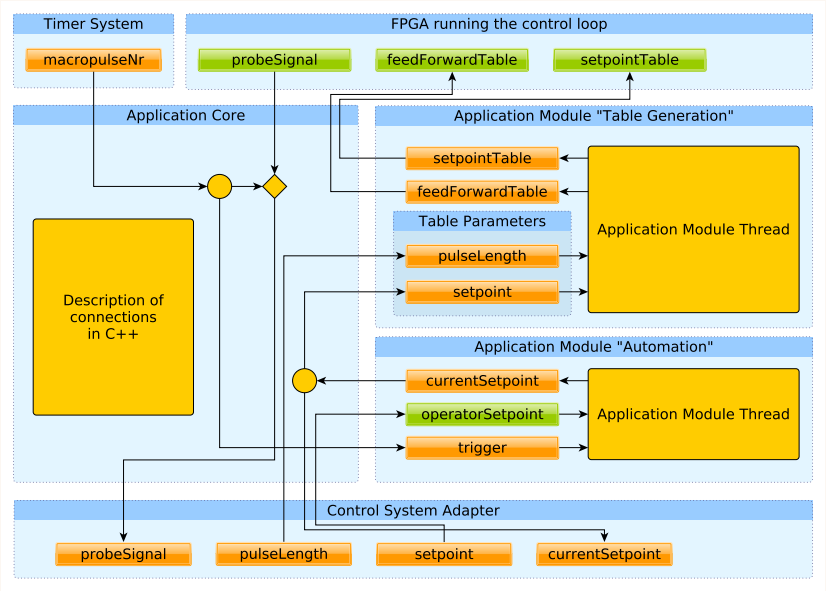
How it all began...

- Kick-off meeting for the OPC-UA adapter with TU-DD (Summer 2016)
- Why do we treat control system variables and registers differently?
- Control system variable $\Rightarrow$ Server
- DeviceAccess register $\Rightarrow$ Client
- Conceptionally the same, can sometimes even be exchanged!
- DOOCS example:
  - Client $\xrightarrow{\text{write via RPC}}$ Server
  - Server $\xrightarrow{\text{send via 0MQ}}$ Client
- Of course one has to decide for one implementation, but no fundamental difference in the application

How it all began...



The structure of ApplicationCore



Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!

Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!
- Separate actual application code (algorithms etc.) from control system and device implementation details

Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!
- Separate actual application code (algorithms etc.) from control system and device implementation details
- Encourage modular applications (even beyond DOOCS locations)

Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!
- Separate actual application code (algorithms etc.) from control system and device implementation details
- Encourage modular applications (even beyond DOOCS locations)
- Clean and simple interface, avoid boiler plate code as much as C++11 allows

Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!
- Separate actual application code (algorithms etc.) from control system and device implementation details
- Encourage modular applications (even beyond DOOCS locations)
- Clean and simple interface, avoid boiler plate code as much as C++11 allows
- Avoid the need for user callback functions (excessive use makes code unreadable)

Central ideas for ApplicationCore

Goal: provide framework for implementing applications which are "naturally" control-system-independent

- If we abstract away differences between control system and device variables, we will less likely make our application sensitive to specific control systems!
- Separate actual application code (algorithms etc.) from control system and device implementation details
- Encourage modular applications (even beyond DOOCS locations)
- Clean and simple interface, avoid boiler plate code as much as C++11 allows
- Avoid the need for user callback functions (excessive use makes code unreadable)
- Allow publishing a device register into the control system with a single code line

Short refresh of C++ 11

- Default arguments to member constructors with braces

```
struct SomeClass {
  std::string myText{"Hello World"};
};
```

Short refresh of C++ 11

- Default arguments to member constructors with braces

```
struct SomeClass {
  std::string myText{"Hello World"};
};
```

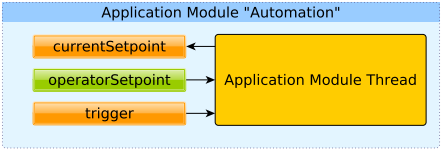
- Brace-initialisers when passing arguments

```
void someFunction(std::vector<int> values);
void main() {
  someFunction({1, 42, 33});
}
```

Central ideas for ApplicationCore

$$\text{Variable} = \text{Register} = \text{Property}$$

Application Module



Application Module (idealised code)

```
struct Automation : public ctk::ApplicationModule {

  ctk::ScalarInput<double> opSP{"opSP", "MV", "Setpoint given by operator"};
  ctk::ScalarOutput<double> curSP{"curSP", "MV", "Automated setpoint"};
  ctk::ScalarInput<int> trigger{"trigger", "", "Macropulse trigger"};









};
```

Application Module (idealised code)

```
struct Automation : public ctk::ApplicationModule {

  ctk::ScalarInput<double> opSP{"opSP", "MV", "Setpoint given by operator"};
  ctk::ScalarOutput<double> curSP{"curSP", "MV", "Automated setpoint"};
  ctk::ScalarInput<int> trigger{"trigger", "", "Macropulse trigger"};

  void mainLoop() {
    while(true) {
      trigger.read();          // wait until next trigger
      opSP.readLatest();       // just get the current value
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();
      }
    }
  }
};
```

Application Module (towards actual code - step 1)

```
struct Automation : public ctk::ApplicationModule {

  ctk::ScalarInput<double> opSP{this, "opSP", "MV", "..."};
  ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "..."};
  ctk::ScalarInput<int> trigger{this, "trigger", "", "..."};

  void mainLoop() {
    while(true) {
      trigger.read();
      opSP.readLatest();
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();
      }
    }
  }
};
```

Application Module (towards actual code - step 1)

```
struct Automation : public ctk::ApplicationModule {

  ctk::ScalarInput<double> opSP{this, "opSP", "MV", "..."};
  ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "..."};
  ctk::ScalarInput<int> trigger{this, "trigger", "", "..."};

  void mainLoop() {
    while(true) {
      trigger.read();
      opSP.readLatest();
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();
      }
    }
  }
};
```

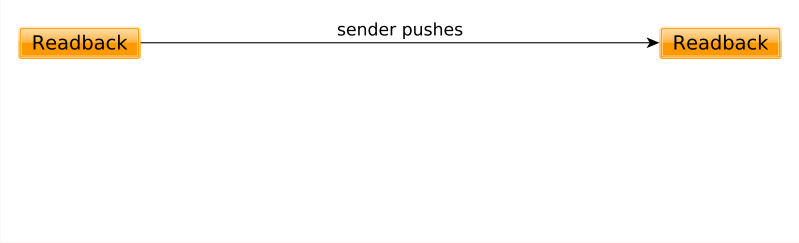
Need to know the owner!

Application Module (towards actual code - step 2)

```
struct Automation : public ctk::ApplicationModule {
  using ctk::ApplicationModule::ApplicationModule;      ⟵ Inherit constructor
  ctk::ScalarInput<double> opSP{this, "opSP", "MV", "..."};
  ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "..."};
  ctk::ScalarInput<int> trigger{this, "trigger", "", "..."};

  void mainLoop() {
    while(true) {
      trigger.read();
      opSP.readLatest();
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();
      }
    }
  }
};
```

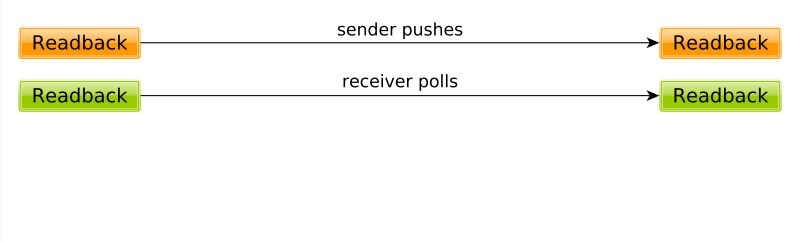
Excursus: Update modes

- Generalise the client/server concept



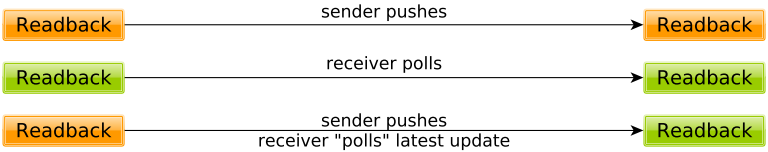
Excursus: Update modes

- Generalise the client/server concept



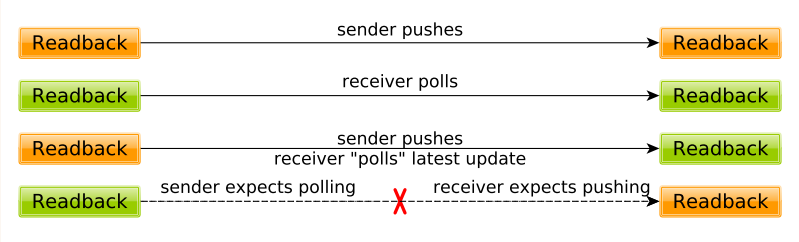
Excursus: Update modes

- Generalise the client/server concept



Excursus: Update modes

- Generalise the client/server concept



Application Module (actual code!)

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

  void mainLoop() {
    while(true) {
      trigger.read();
      opSP.readLatest();     // opSP.read() would be equivalent
      if(std::abs(opSP - curSP) > 0.01) {
        curSP += std::min( std::max(opSP - curSP, 0.1), -0.1);
        curSP.write();
      }
    }
  }
};
```

Application Module (actual code!)

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

  vo

  }
};
```

Interface from DeviceAccess

- ScalarPollInput / ScalarPushInput / ScalarOutput
  ⇒ ScalarRegisterAccessor
- ArrayPollInput / ArrayPushInput / ArrayOutput
  ⇒ OneDRegisterAccessor

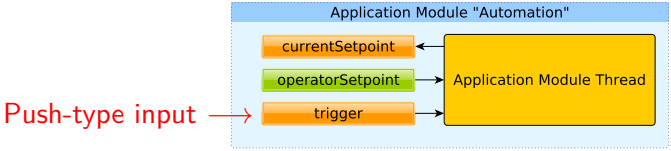
Application Module (actual code!)

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

  vo
  }
};
```

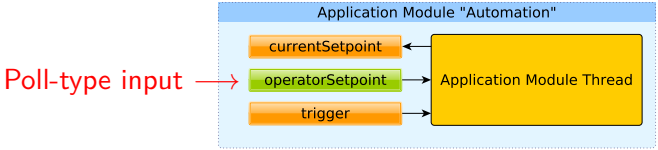
Interface from DeviceAccess

- ScalarPollInput / ScalarPushInput / ScalarOutput
  ⇒ ScalarRegisterAccessor
- ArrayPollInput / ArrayPushInput / ArrayOutput
  ⇒ OneDRegisterAccessor
- Actual inheritance!
- Only adds *inversion of control*

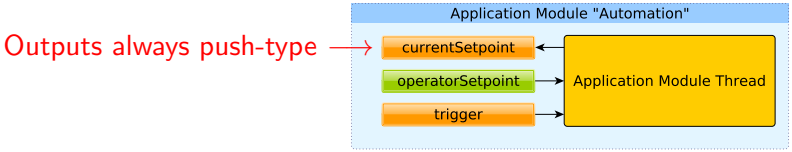
Application Module



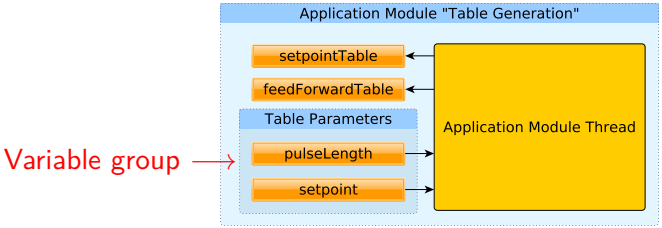
Application Module



Application Module



Group your variables



Group your variables

- An ApplicationModule can contain variables — and groups of variables

```
struct TableGeneration : public ctk::ApplicationModule {
  // ...
  struct TableParameters : public ctk::VariableGroup {
    using ctk::VariableGroup::VariableGroup;
    ctk::ScalarPushInput<double> pulseLength{this, "pulseLength", "us", "..."};
    ctk::ScalarPushInput<double> setpoint{this, "setpoint", "MV", "..."};
  };
  TableParameters tableParams{this, "tableParameters", "..."};









};
```

Group your variables

- An ApplicationModule can contain variables — and groups of variables

```
struct TableGeneration : public ctk::ApplicationModule {
  // ...
  struct TableParameters : public ctk::VariableGroup {
    using ctk::VariableGroup::VariableGroup;
    ctk::ScalarPushInput<double> pulseLength{this, "pulseLength", "us", "..."};
    ctk::ScalarPushInput<double> setpoint{this, "setpoint", "MV", "..."};
  };
  TableParameters tableParams{this, "tableParameters", "..."};

  void mainLoop() {
    while(true) {
      // ...
      tableParams.readAll();    // blocks until *all* variables are changed
    }
  }
};
```

Group your variables

- An ApplicationModule can contain variables — and groups of variables

```
struct TableGeneration : public ctk::ApplicationModule {
  // ...
  struct TableParameters : public ctk::VariableGroup {
    using ctk::VariableGroup::VariableGroup;
    ctk::ScalarPushInput<double> pulseLength{this, "pulseLength", "us", "..."};
    ctk::ScalarPushInput<double> setpoint{this, "setpoint", "MV", "..."};
  };
  TableParameters tableParams{this, "tableParameters", "..."};

  void mainLoop() {
    while(true) {
      // ...
      tableParams.readAny();    // block until *any* variable is changed
    }
  }
};
```

Defining the full application

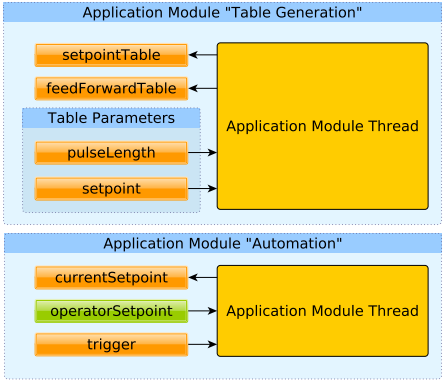
Most code will go into those mainLoop() implementations!

Defining the full application

Most code will go into those mainLoop() implementations!

But some quite complicated concepts are still missing...

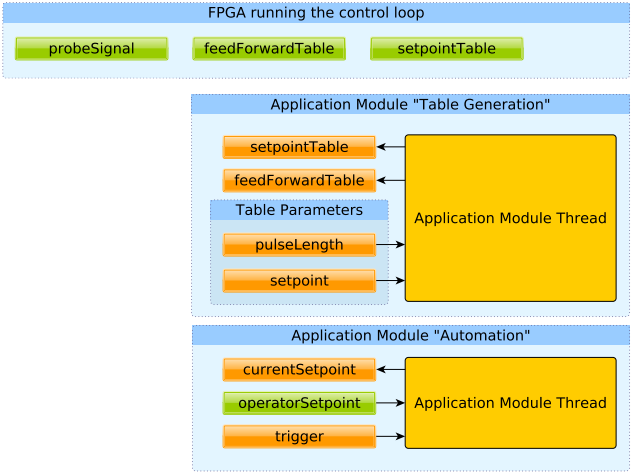
Defining the full application



Defining the full application (partial actual code)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("demoApp") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};









};
```

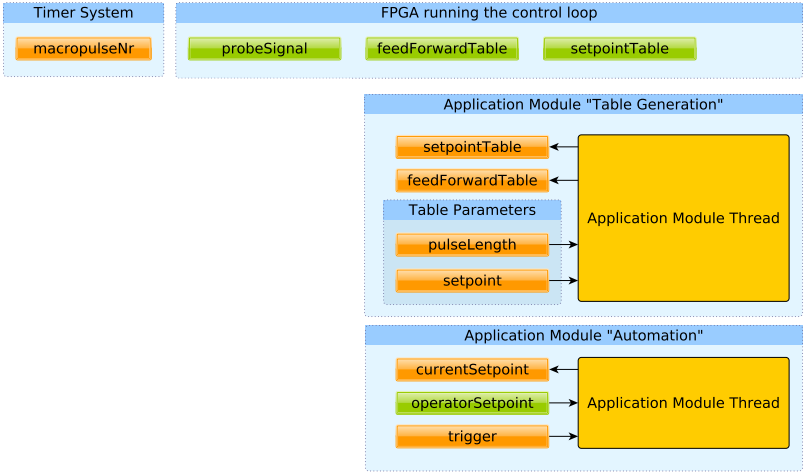
Defining the full application



Defining the full application (partial actual code)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("demoApp") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};
    ctk::DeviceModule dev{"Device0"};




};
```

Defining the full application



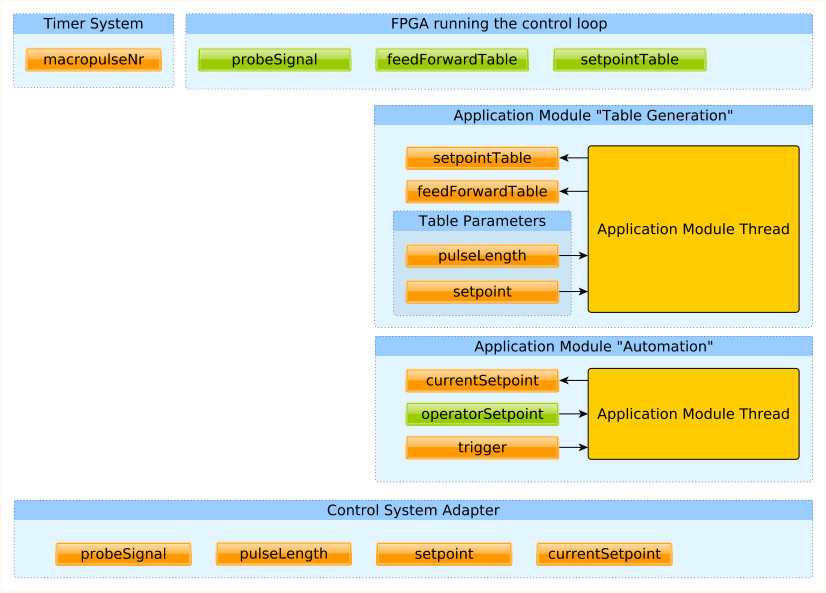
Defining the full application (partial actual code)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("demoApp") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};
    ctk::DeviceModule dev{"Device0"};
    ctk::DeviceModule timer{"Timer"};




};
```

The .dmap file

```
@LOAD_LIB /usr/lib/libChimeraTK-DeviceAccess-DoocsBackend.so
ADC0      sdm://./pci:pcieunis4                    sincav_sis8300l_hzdr_srf_r2102.map
Device0   sdm://./logicalNameMap                   llrfserver.xlmap
Timer     sdm://./doocs=TEST.DOOCS,TIMER2_SRV,MTCACPU.0      none
```

Defining the full application



Defining the full application (partial actual code)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("llrfServer") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};
    ctk::DeviceModule dev{"Device0"};
    ctk::DeviceModule timer{"Timer"};
    ctk::ControlSystemModule cs;



};
```

Defining the full application (partial actual code)

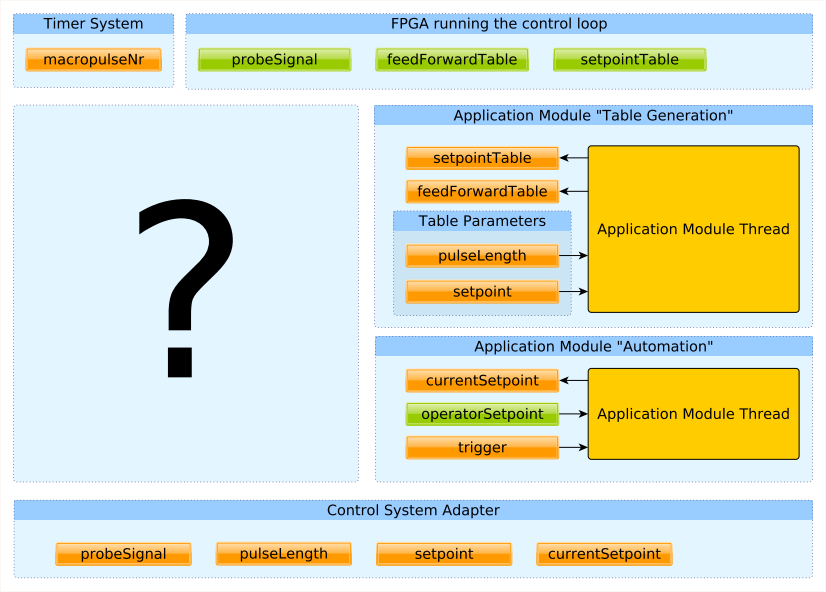
```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("llrfServer") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};
    ctk::DeviceModule dev{"Device0"};
    ctk::DeviceModule timer{"Timer"};
    ctk::ControlSystemModule cs;



};
LLRFServer theLLRFServer;
```

Defining the full application (partial actual code)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("llrfServer") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation"  ...
    TableGeneration tableGeneration{thi...eneration", "..."};
    ctk::DeviceModule dev{"Devic...
    ctk::DeviceModule time...
    ctk::ControlSys...s;

};
LLRFServer theLLRFServer;
```

No main() function shall be defined, it is coming from the adapter!

Defining the full application



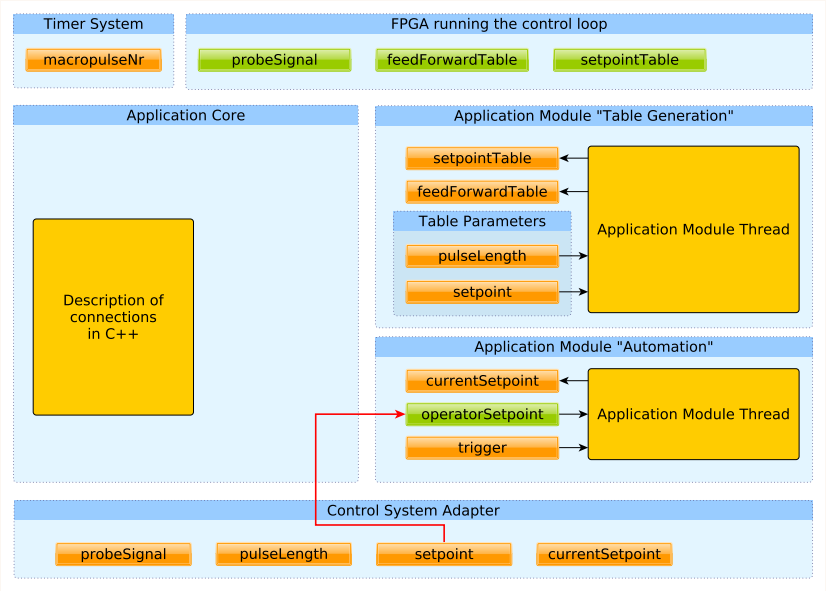
Defining the full application (actual code!)

```
struct LLRFServer : public ctk::Application {
    LLRFServer() : Application("llrfServer") {}
    ~LLRFServer() { shutdown(); }

    Automation automation{this, "automation", "..."};
    TableGeneration tableGeneration{this, "tableGeneration", "..."};
    ctk::DeviceModule dev{"Device0"};
    ctk::DeviceModule timer{"Timer"};
    ctk::ControlSystemModule cs;

    void defineConnections();

};
LLRFServer theLLRFServer;
```

Defining the connections (partial code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");









}
```

Defining the connections



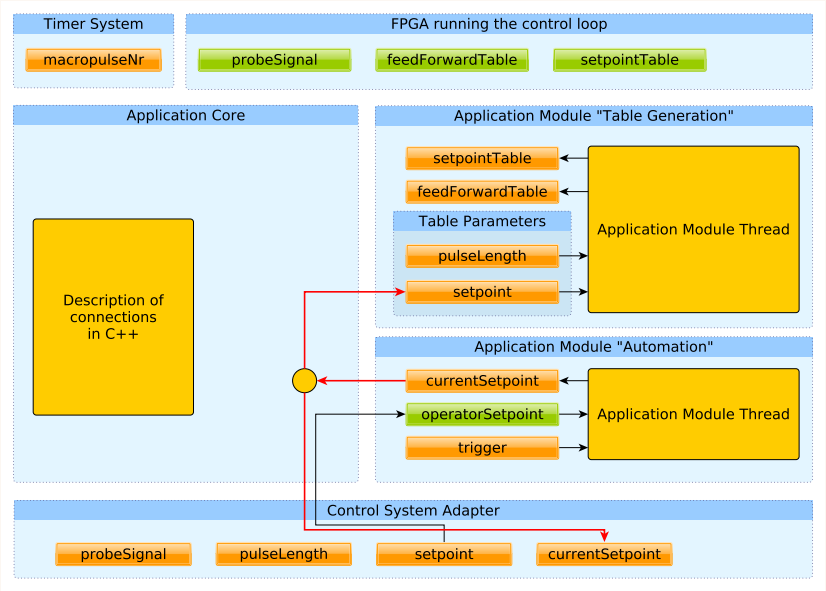
Defining the connections (partial code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;









}
```

Defining the connections (partial code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;

    ↑
    This will cause the creation of a ProcessVariable in the ControlSystemAdapter

}
```

Defining the connections



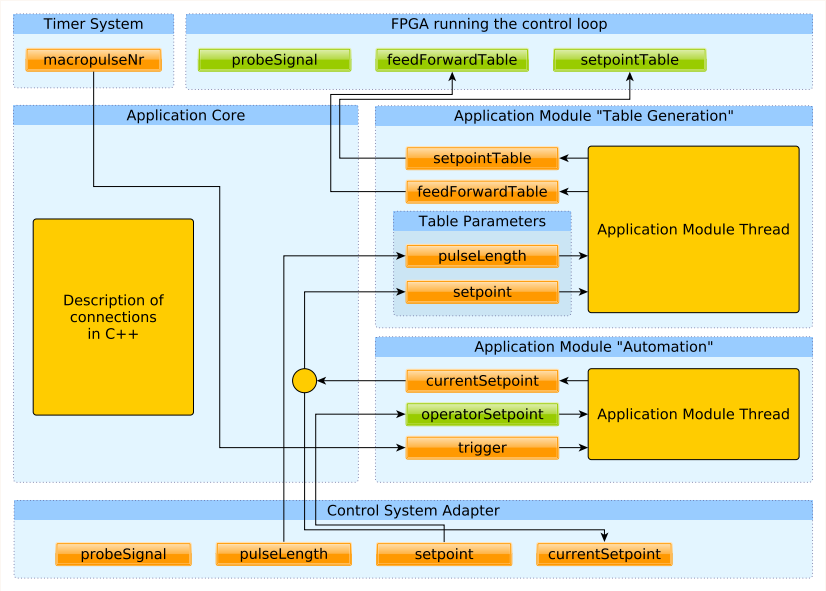
Defining the connections (partial code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");









}
```

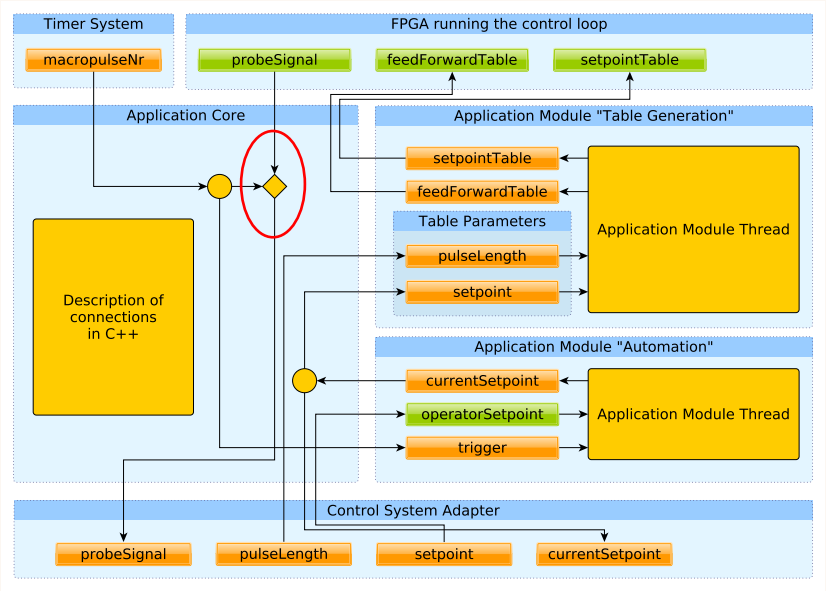
Defining the connections (partial idealised code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");


  timer("macropulseNr") >> automation.trigger;

  cs("pulseLength") >> tableGeneration.tableParams.pulseLength;

  tableGeneration.setpointTable >> dev("setpointTable");
  tableGeneration.feedforwardTable >> dev("feedforwardTable");



}
```

Defining the connections

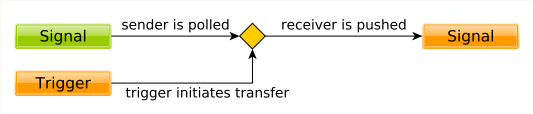


Defining the connections



Triggers

- When connecting a poll-type output (e.g. a PCIe register) with a push-type input, a trigger for the transfer is needed



- Any push-type variable can be used as trigger, its value will be ignored

Defining the connections (idealised code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");


  timer("macropulseNr") >> automation.trigger;

  cs("pulseLength") >> tableGeneration.tableParams.pulseLength;

  tableGeneration.setpointTable >> dev("setpointTable");
  tableGeneration.feedforwardTable >> dev("feedforwardTable");

  dev("probeSignal") [ timer("macropulseNr") ] >> cs("probeSignal");

}
```

Defining the connections (idealised code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");
  tableGeneration.setpointTable >> dev("setpointTable");
  tableGeneration.feedforwardTable >> dev("feedforwardTable");

→ dev("probeSignal") [ timer("macropulseNr") ] >> cs("probeSignal");

}
```

- Neither DeviceModule nor ControlSystemModule define their variable types
- DeviceAccess allows to read a register as any type
- ControlSystemAdater variables are created on-the-fly
- We need to specify the type and array length here!

Defining the connections (idealised code)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");
  tableGeneration.setpointTable >> dev("setpointTable");
  tableGeneration.feedforwardTable >> dev("feedforwardTable");

→ dev("probeSignal", typeid(int), 16384) [ timer("macropulseNr") ]
    >> cs("probeSignal");
}
```

- Neither DeviceModule nor ControlSystemModule define their variable types
- DeviceAccess allows to read a register as any type
- ControlSystemAdater variables are created on-the-fly
- We need to specify the type and array length here!

Defining the connections (actual code!)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");


→ timer("macropulseNr") >> automation.trigger;

  cs("pulseLength") >> tableGeneration.tableParams.pulseLength;

}
```

- Similar problem here, but this time the UpdateMode is wrong
- DeviceAccess variables are by default poll-type

Defining the connections (actual code!)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");

  auto macropulseNr = timer("macropulseNr", typeid(int), 1, ctk::UpdateMode::push);
  macropulseNr >> automation.trigger;

  cs("pulseLength") >> tableGeneration.tableParams.pulseLength;

}
```

- Similar problem here, but this time the UpdateMode is wrong
- DeviceAccess variables are by default poll-type
- Need to change this to push-type
- DOOCS backend: push-type means ZeroMQ subscription!

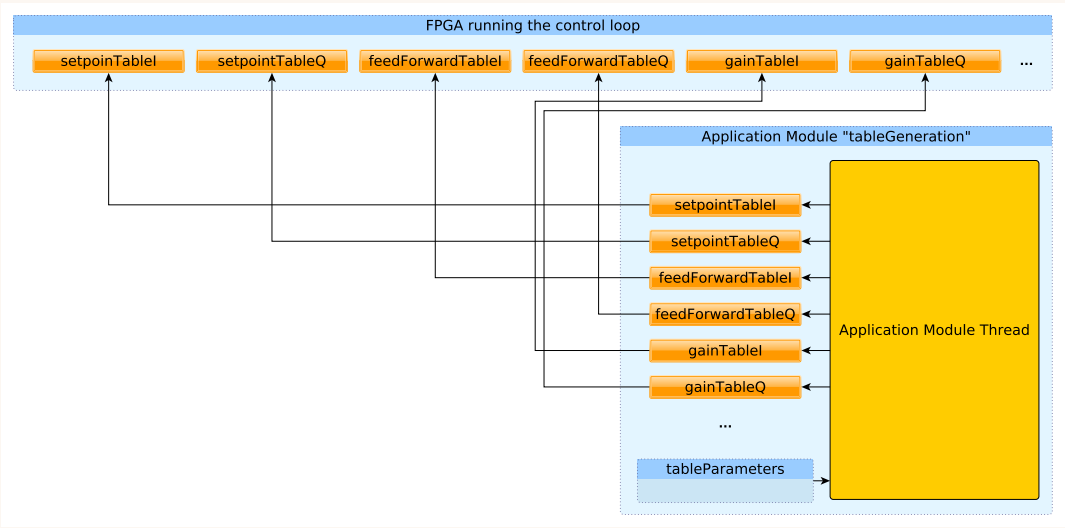
Defining the connections (actual code!)

```
void LLRFServer::defineConnections() {
  mtca4u::setDMapFilePath("deviceMapFile.dmap");

  cs("setpoint") >> automation.opSP;
  automation.curSP >> tableGeneration.tableParams.setpoint >> cs("currentSetpoint");

  auto macropulseNr = timer("macropulseNr", typeid(int), 1, ctk::UpdateMode::push);
  macropulseNr >> automation.trigger;

  cs("pulseLength") >> tableGeneration.tableParams.pulseLength;

  tableGeneration.setpointTable >> dev("setpointTable");
  tableGeneration.feedforwardTable >> dev("feedforwardTable");

  dev("probeSignal", typeid(int), 16384) [ macropulseNr ] >> cs("probeSignal");

}
```

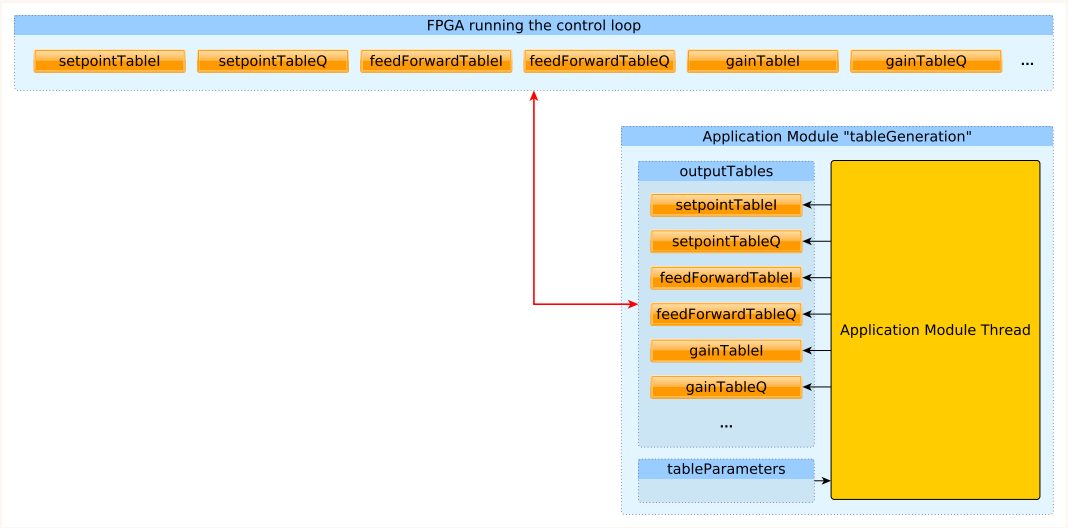
Too much code!

This would be way too much code for a big server.
One line of connection code per variable/register!

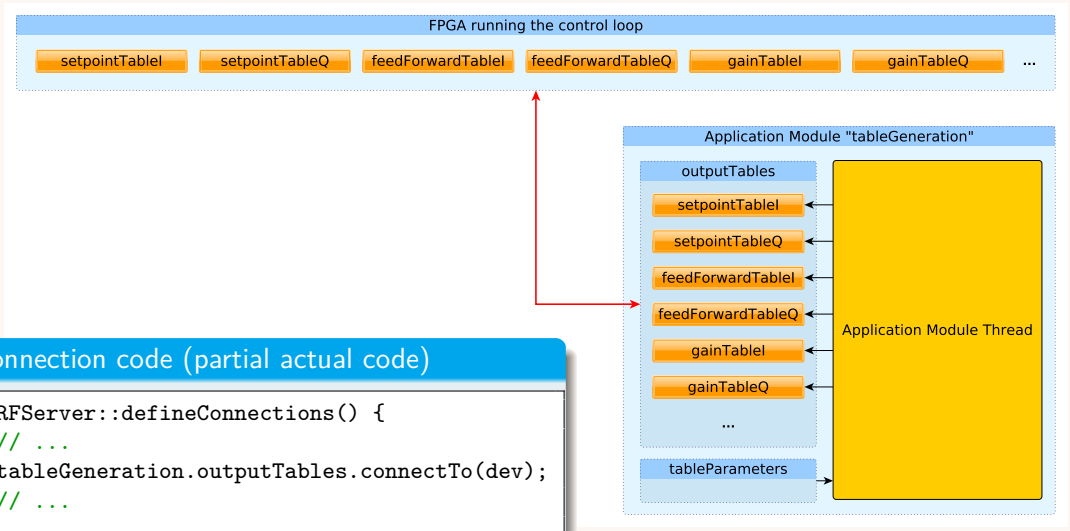
Group connections



Group connections



Group connections



Connection code (partial actual code)

```
LLRFServer::defineConnections() {
  // ...
  tableGeneration.outputTables.connectTo(dev);
  // ...
}
```

connectTo()

- Connect all variables of module with variables of same name in other module
- Works the same way with VariableGroup

connectTo()

- Connect all variables of module with variables of same name in other module
- Works the same way with VariableGroup
- Does not care about additional variables in the target module
- Recurses into sub-groups

connectTo()

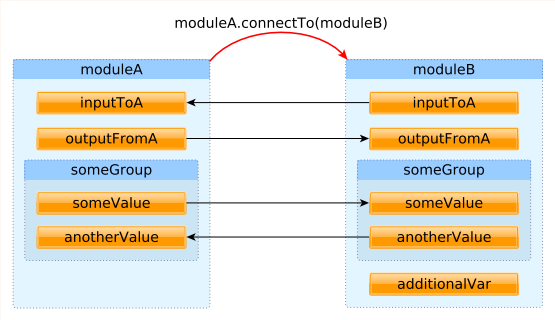
- Connect all variables of module with variables of same name in other module
- Works the same way with VariableGroup
- Does not care about additional variables in the target module
- Recurses into sub-groups
- Works nicely with ControlSystemModule which creates variables on demand

connectTo()

- Connect all variables of module with variables of same name in other module
- Works the same way with VariableGroup
- Does not care about additional variables in the target module
- Recurses into sub-groups
- Works nicely with ControlSystemModule which creates variables on demand
- Even acts on inputs and outputs simultaneously

connectTo()

- Connect all variables of module with variables of same name in other module
- Works the same way with VariableGroup
- Does not care about additional variables in the target module
- Recurses into sub-groups
- Works nicely with ControlSystemModule which creates variables on demand
- Even acts on inputs and outputs simultaneously



Model your information

- Build deep hierarchies by using VariableGroup and ModuleGroup

Model your information

- Build deep hierarchies by using VariableGroup and ModuleGroup
- Add tags to variables (even to entire modules/groups)

Declaration of variable in module:

```
ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "...", {"TableGen"}};
```

Model your information

- Build deep hierarchies by using VariableGroup and ModuleGroup
- Add tags to variables (even to entire modules/groups)

  Declaration of variable in module:

```
ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "...", {"TableGen"}};
```

- Search for tag and connect by tags

  Connection code:

```
automation.findTag("TableGen").connectTo(tableGeneration);
```

Model your information

- Build deep hierarchies by using VariableGroup and ModuleGroup
- Add tags to variables (even to entire modules/groups)

  Declaration of variable in module:

```
ctk::ScalarOutput<double> curSP{this, "curSP", "MV", "...", {"TableGen"}};
```

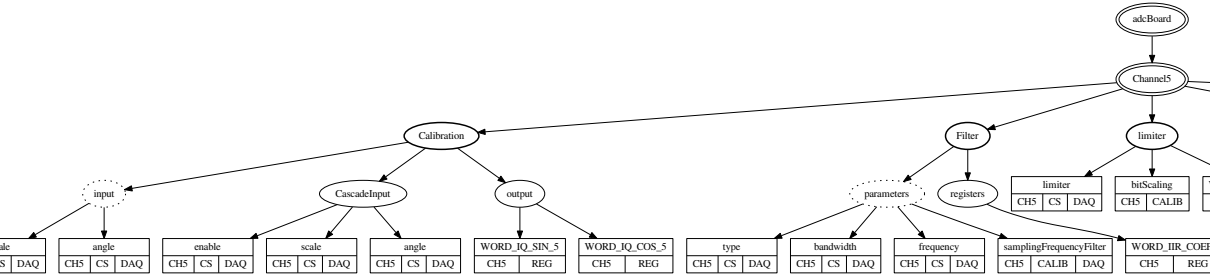
- Search for tag and connect by tags

  Connection code:

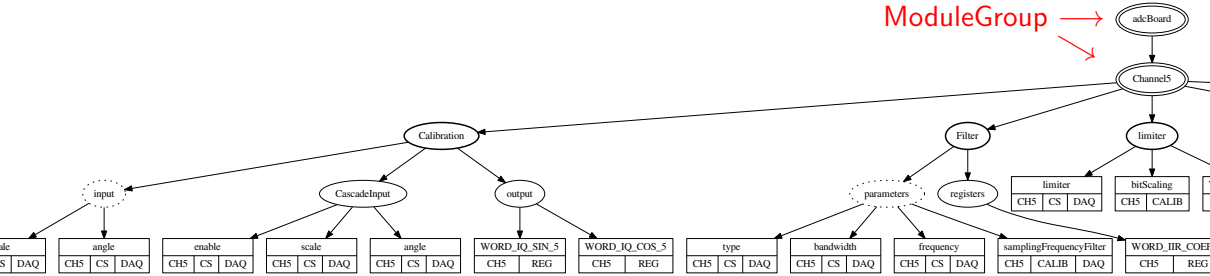
```
automation.findTag("TableGen").connectTo(tableGeneration);
```

- Variables can have more than one tag
- findTag() can be nested (e.g. `someModule.findTag("tagA").findTag("tagB")`)

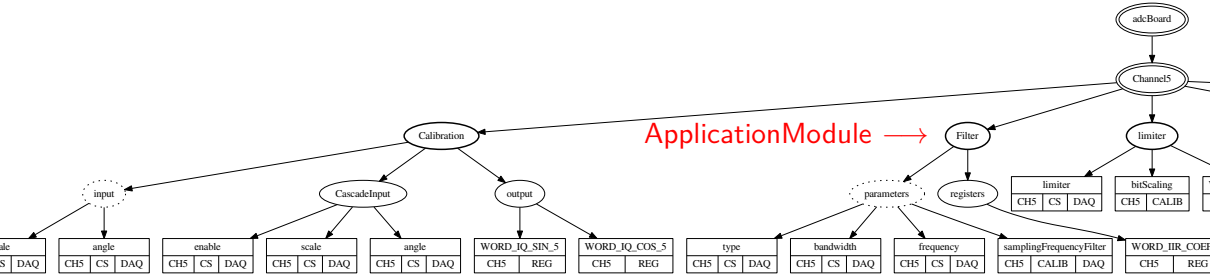
Example: LLRF server module adcBoard, Channel 5 (partial)



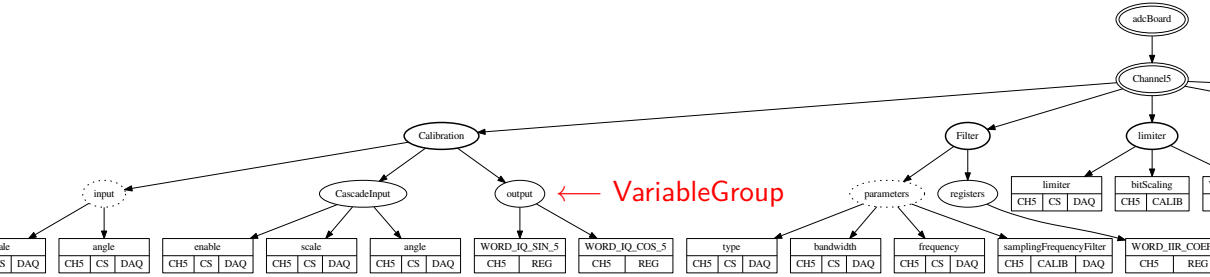
Example: LLRF server module adcBoard, Channel 5 (partial)



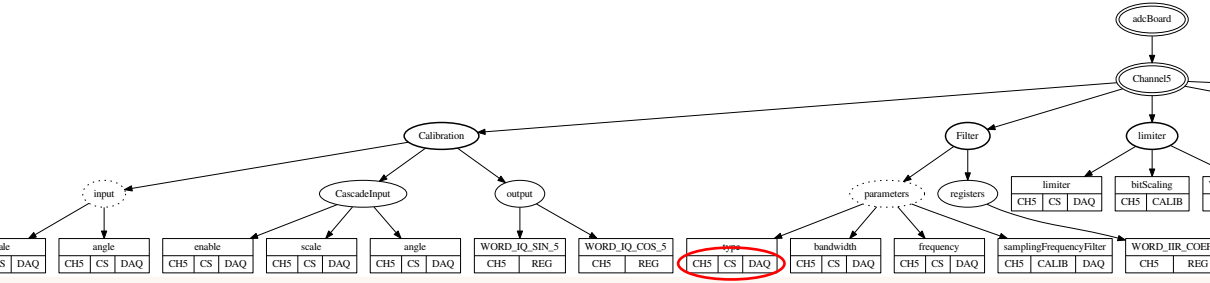
Example: LLRF server module adcBoard, Channel 5 (partial)



Example: LLRF server module adcBoard, Channel 5 (partial)



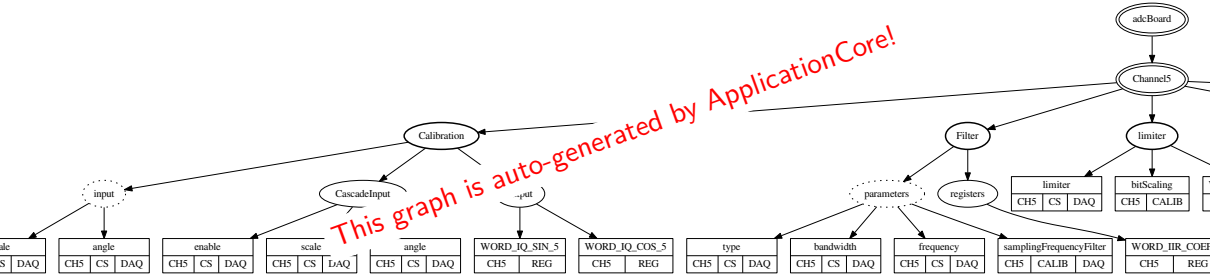
Example: LLRF server module adcBoard, Channel 5 (partial)



Tags

- Tag “CH5”: other modules also provide variables for Channel 5
- Tags “REG” and “CS”: device registers and control system variables
- Tag “CALIB”: global calibration values
- Tag “DAQ”: variables go into the internal DAQ system (if enabled)

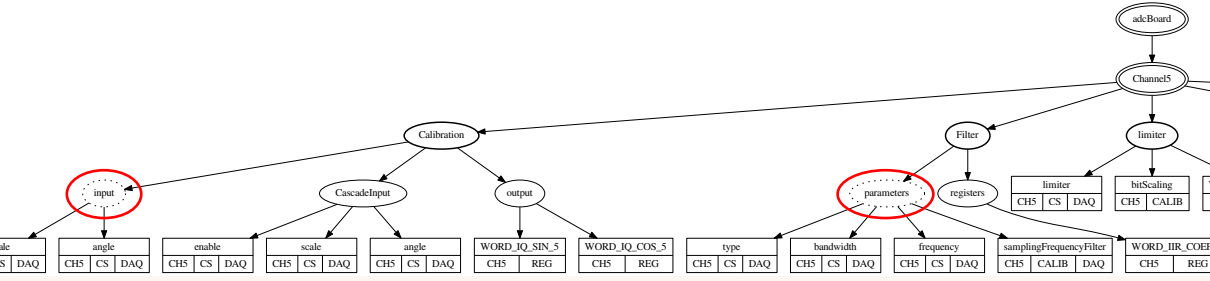
Example: LLRF server module adcBoard, Channel 5 (partial)



Tweak the model

In constructor of your ModuleGroup etc.:

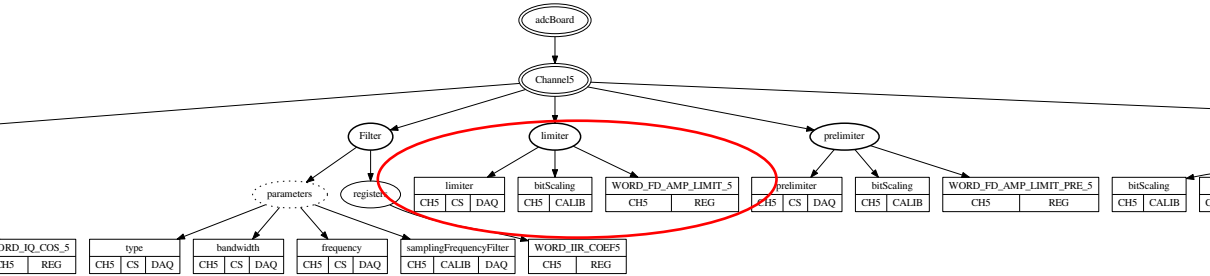
- Eliminate hierarchy levels (`Module::setEliminateHierarchy()`)



Tweak the model

In constructor of your ModuleGroup etc.:

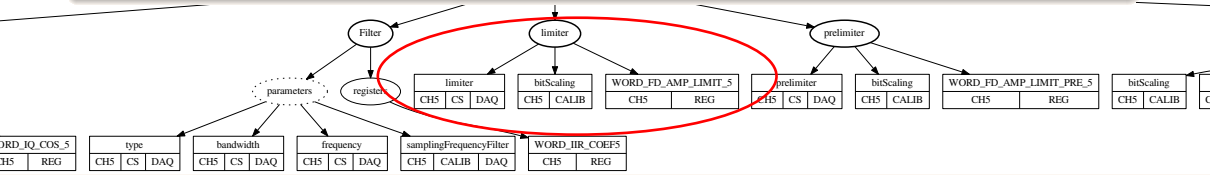
- Eliminate hierarchy levels (`Module::setEliminateHierarchy()`)
- Override variable names etc. in generic modules (`ScalarOutput::setMetaData()`)



Tweak the model

Generic module

```
template<typename T>
struct MultiplierModule : public ctk::ApplicationModule {
  using ctk::ApplicationModule::ApplicationModule;

  ctk::ScalarPushInput<T> input{this, "input", "", "Input value to be scaled"};
  ctk::ScalarPushInput<T> factor{this, "factor", "", "Scaling factor"};
  ctk::ScalarOutput<T> output{this, "output", "", "Output value after scaling"};

  void mainLoop();
};
```



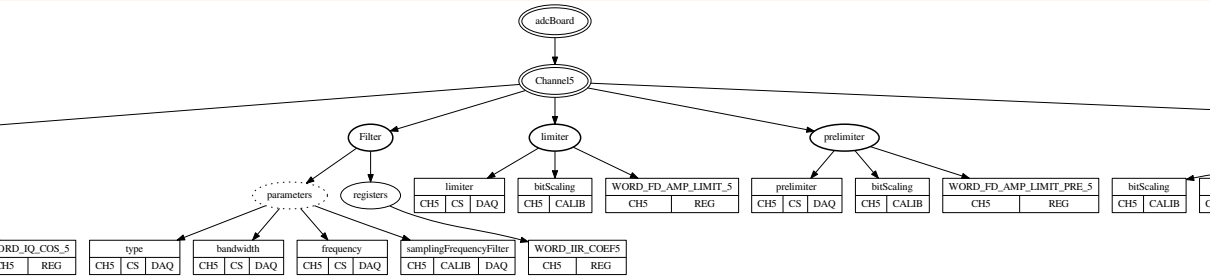
Tweak the model

In constructor of your ModuleGroup etc.:

- Eliminate hierarchy levels (`Module::setEliminateHierarchy()`)
- Override variable names etc. in generic modules (`ScalarOutput::setMetaData()`)

In your `defineConnections()`:

- Eliminate all hierarchy levels (`Module::flatten()`)



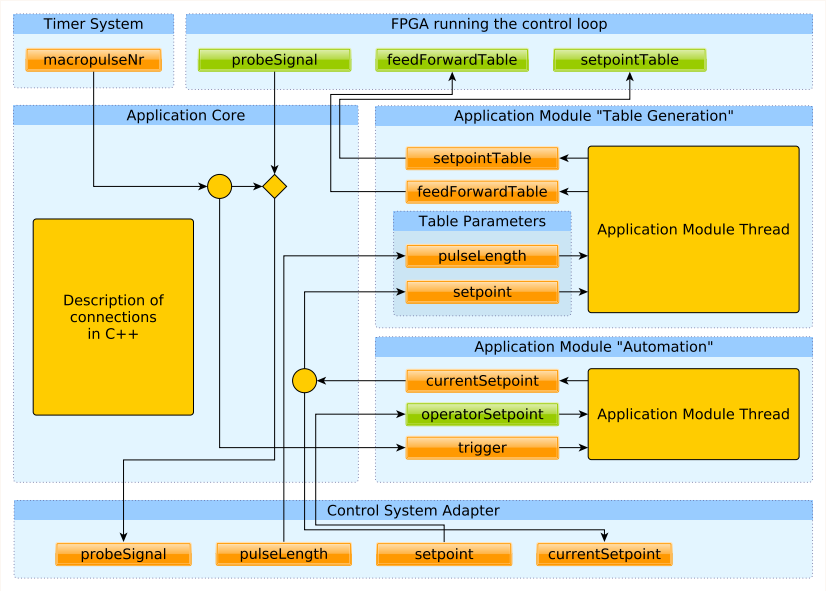
Arrays of modules

```
struct Channel : public ctk::ApplicationModule {
    // ...
};
struct AdcBoard : public ctk::ModuleGroup {
    // ...
    std::vector<Channel> channels;
    // ...
};
```

```
AdcBoard::AdcBoard(/* ... */) {
    for(size_t i = 0; i < numberOfChannels; ++i) {
      channels.emplace_back(this, "Channel"+std::to_string(i), /*...*/);
    }
}
```

- Similarly possible with variables (put e.g. `ctk::ScalarOuput` into `std::vector`)

The structure of ApplicationCore



Servers currently in development using ApplicationCore

- LLRF ctrl server: single cavity CW version for HZDR and CMTB
  - successfull tests at HZDR last week with full WinCC / OPC UA integration
  - long-term plan: extend for FLASH/XFEL and replace pure DOOCS server
- FRED server
- Watchdog for HZDR

How to get started

- Demo app available! See source code: `example/demoApp.cc`
- Will be reworked soon with the example from this talk

How to get started

- Demo app available! See source code: `example/demoApp.cc`
- Will be reworked soon with the example from this talk
- Simple applications don't need fancy information models
- Just put all control system variables into one group, all device accessors into another (and use `connectTo()`)

How to get started

- Demo app available! See source code: `example/demoApp.cc`
- Will be reworked soon with the example from this talk
- Simple applications don't need fancy information models
- Just put all control system variables into one group, all device accessors into another (and use `connectTo()`)
- github: `https://github.com/ChimeraTK/ApplicationCore`
- Documentation (work in progress): `https://chimeratk.github.io`

How to get started

- Demo app available! See source code: `example/demoApp.cc`
- Will be reworked soon with the example from this talk
- Simple applications don't need fancy information models
- Just put all control system variables into one group, all device accessors into another (and use `connectTo()`)
- github: `https://github.com/ChimeraTK/ApplicationCore`
- Documentation (work in progress): `https://chimeratk.github.io`
- Debian packages available for Ubuntu 16.04 (public DOOCS repository)

(backup)

Planned improvement (long term, not complete)

- Add types: bool and void (for triggers)
- "Blind" variables: add variable to information model, e.g. to publish a devie register to the control system despite not being used by the app
- "Tiny" application modules: save extra thread / context switches for small operation (e.g. just scaling a value)
- For documentation etc.: create more graphs showing the application structure in different views (actual realisation, ...)
- Enforce data consistency between variables in transfers
- Create library of generic modules (like the MultiplierModule)
- Allow defining the connections by XML file