DAQ 2 SoC communication library: status of the prototype

Andrei Kazarov

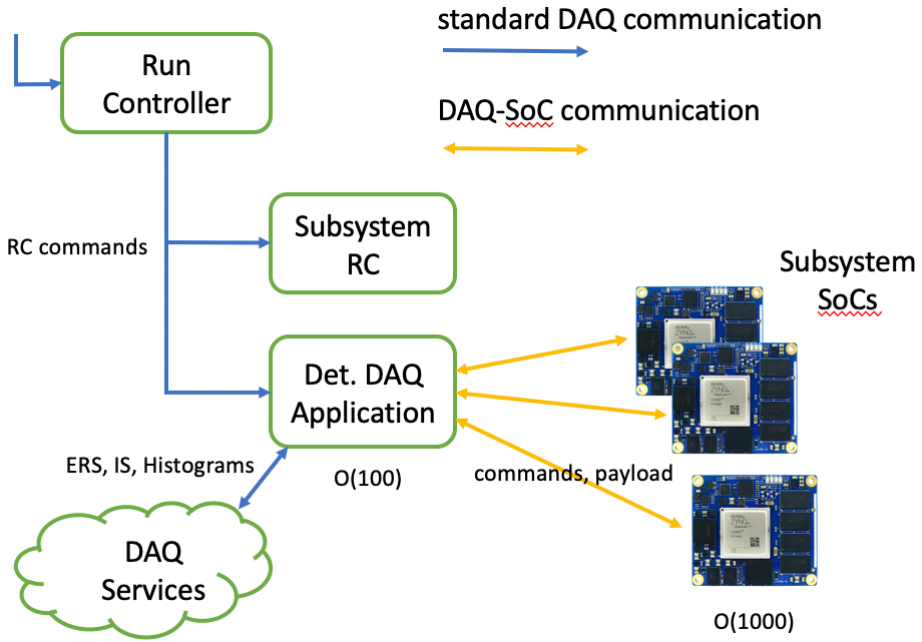
University of Johannesburg, SA

Functionality overview and high-level design

- A protocol or an interface to communicate commands (and more generally, to **exchange information**) to a SoC system: SoC-DAQ Interface (**SoC-DAQI**)
- An DAQ application serves as a gateway from SoC eco-system to the DAQ services

Examples:

- DAQ application regularly gets a status from SoC and publishes it in DAQ IS, or in case of an reported error, issues an ERS message
- DAQ application receives Run Control transition command from its parent, distributes it (when necessary) to the controlled SoC systems
- DAQ application passes configuration data (e.g. a JSON string) to SoC
- DAQ application gets data from SoC and creates and publishes histograms in DAQ

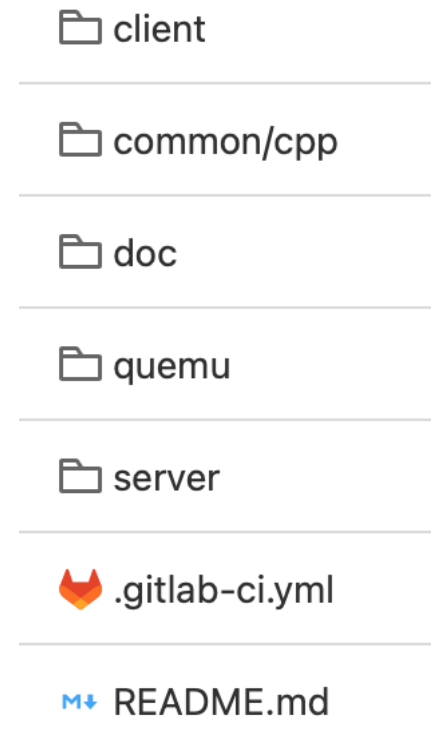


Prototype: a HTTP based, nginx user module

- use **HTTP** (s) as transport layer, standard POST and GET requests
  - payload can be passed as part of request and returned back
- use of **nginx** http server (pre-built binary) as server-side application, handling all networking, connection, threading functionality
- user code is executed in a request handler context
  - loaded from a .so library compiled from a Makefile: user implements a function of a class with a predefined signature
- client-side: any HTTP client (a helper library available in C++)
- payload (can be passed as part of request and returned back): JSON string, handy conversion to and from standard C++ objects and containers (header only)
- **no dependencies** from TDAQ, LCG, only Boost (C++ client side) and C++11
- Prototype is in gitlab
  - https://gitlab.cern.ch/akazarov/daq2soc
  - https://daq2soc.docs.cern.ch/index.html

Package content https://gitlab.cern.ch/akazarov/daq2soc

- client: C++ API files: header file, implementation, example and a Makefile to compile it
- server:
  - nginx-module: sources and Makefile for compiling a user code into a dynamic daq2soc library loaded by nginx at runtime
  - nginx-server: precompiled binaries for aarch64 and x86_64 nginx http server and dynamic module
- common: a single-header file for JSON 2 C++ conversions: used in both client and server for parsing the payload
- quemu: how to run aarch64 CC8 linux on x86_64 host and test the binaries



Client-side API

- There are two classes exposed to user: Sender and AsyncHandler
- Payload is a vector of bytes

```
#include <daqsoc.hpp>
```

Public Member Functions

	Sender (const std::string &host) Constructor for **Sender** class. More...
Data	**SendCommandSync** (const std::string &endpoint, const std::map< std::string, std::string > &parameters, const **Payload** &data_in) Syncronously sends a command to server and waits for a result. More...
std::shared_ptr< **AsyncHandler** >	**SendCommandAsync** (const std::string &endpoint, const std::map< std::string, std::string > &parameters, const **Payload** &data_in) Asyncronously sends a command to server, returning immediately a handle to ayncronous result. More...

Server-side API

- User need to implement UserData::daq_request_function and process the client request (synchronously to the client call)
  - you can have some payload passed in and can prepare a payload to return
  - UserData class holds user attributes persisting across requests
- a dedicated thread_function allows to implement code and data structures which run independently on the user requests

	UserData () Constructor. Initialize your attributes here. More...
virtual std::tuple< int, std::vector< uint8_t > >	**daq_request_function** (const std::string &endpoint, const std::map< std::string, std::string > &parameters, const std::vector< uint8_t > &data_in) User function to process a request received from daq2soc client. More...
virtual void	**thread_function** (int &stop) User function to execute a code in separate thread. More...

API: C++ to JSON serialization (and back to C++)

template<typename ... Args>

std::string **data2json** (Args &&... args)

Function to serialize a tuple-like C++ data into a JSON string. More...

template<typename ... Args>

std::string **data2json** (std::tuple< Args... > tuple)

A wrapper for converting a tuple<Args...> into JSON. More...

template<typename... DataObjects>

auto **json2data** (const std::string_view &json)

A helper function to deserialize a JSON string into a tuple-like C++ data: a counter-part for data2json.

- Serialized data (string) then can be used in Sender to send data to server where it can be parsed back directly into C++ structures
- Example: https://daq2soc.docs.cern.ch/test_json_8cpp-example.html

Example: returning a Histogram

client

```
tdaq::soc::Sender mysender { std::string(host) } ;

tdaq::soc::Payload payload { 'a', 'b', 'c' } ;

std::map<std::string, std::string> parameters { {"partition", "ATLAS"} } ;

tdaq::soc::Data res = mysender.SendCommandSync(“get_histogram”, parameters, payload) auto
const& data = std::get<1>(res) ;

std::string myhist(data.begin(), data.end()) ;

Histogram<10> hist = tdaq::daq2soc::json2data<Histogram<10>>(myhist) ;
```

server

```
example::Histogram<10> hist { "random historgram", {} } ;

// fill with random numbers

std::for_each(std::begin(std::get<1>(hist)), std::end(std::get<1>(hist)), [](float&
x) { x = (float)std::rand()/RAND_MAX ;}) ;

// convert to json std::string json = tdaq::daq2soc::data2json(hist) ;
std::vector<uint8_t> ret_data(std::begin(json), std::end(json)) ;

std::get<1>(ret) = ret_data ;
```

Conclusions

- Prototype is ready
- Waiting for user feedback