QC-Toolkit

Version 0.2 - DRAFT

RWTH Aachen University

April 13, 2015

Abstract

This document describes the current status of the requirement analysis and the implementation progress of the QC-Toolkit project. This project consists in the restructuring of the pulsecontrol (special measure) project of Prof. Dr. Hendrik Bluhm. In short, the QC-Toolkit should increase the modularity and flexibility of pulsecontrol.

Contents

1 Involved People

The Bluhm-Group (Physics)

Prof. Dr. Hendrik Bluhm Project initiator and supervisor.

Patrick Bethke PhD-Student

Pascal Cerfontaine PhD-Student

Tim Botzem PhD-Student

Simon Humpohl Scientific assistant

The Rumpe-Group (Computer Science)

Prof. Dr. Bernhardt Rumpe Project initiator and supervisor.

Deni Raco Student assistant supervisor

Lukas Prediger Scientific assistant

Jerome Bergmann Scientific assistant

2 The Current State

Special measure is a general purpose data acquisition package for (table top scale) physics experiments. Out of that project, the Bluhm-Group created pulsecontrol, a package for defining and tracking AWG pulses for quantum information experiments. It communicates with special-measure in a few places but is largely independent.

As the development of this project took place over several years, pulsecontrol is now a feature rich toolkit. Special measure was written imperatively, so is pulsecontrol. Currently, the project has reached a state where it is hard to understand, modify and repair. To address this issues, the QC-Toolkit Project has been initiated, aiming for a object oriented implementation of pulsecontrol and subsequently adding more functionality with higher abstraction.

2.1 The Driver Interface

Recently, the pulsecontrol toolkit has been expanded by a Multi-AWG concept. This allows to map the channels of a virtual AWG onto hardware channels provided by several AWGs. There are already drivers for all the tools the Bluhm Group written in C++. Currently, we are testing their correctness. If this process is finished, we could directly use them in the QC-Toolkit. As our focus is modularity, we designed the interface of the driver layer as versatile as possible.

2.2 The Pulse Representation in Quantum Mechanics

todo : is this correct? for example, the actual implementation of an algorithm may use pulses as a tool but the algorithm itself is independent from pulses. this sections needs some clarification in my opinion.

In quantum mechanics, a pulse can be described on different layers. The last three layers are part of pulsecontrol:

Algorithm: Description in Quantum-Circuit-Form, the most general form, only theoretical

Optimized Algorithm: The Algorithm can be adapted to the actual setup with error correction and setup relevant environment variables can be set. This layer can also be represented in Quantum-Circuit-Form. One could also make here a separation between the logic gates and the manipulation of environment variables.

PulseGroup: The logic gates can be represented as a PulseGroup, which may contain PulseTabs and Pulses.

PulseTab: Tabular description of a Pulse. Only major supporting points needed. Between two supporting points, a linear interpolation occurs.

Pulse: A table containing in the first row all time points and on the other rows the amplitude for each channel.

3 The Desired State

Until now, we were able to formulate the following requirements for the QC-Toolkit.

Functional

- The system should allow to declare pulses as pulse tables, i.e., certain defining points that are then used to obtain values in between by (linear) interpolation).
- It should be possible to nest pulses, i.e., construct pulses by combining already existing ones.
- A pulse should be able to accept parameters and functions, for reusability.
- Productive pulses shall be stored in a pulse database.
- A concrete pulse used in a certain experiment must be reconstructable, even if the corresponding pulse in the database was changed afterwards.
- The system should allow feedback loops on pulse execution and post process the measured data to enable easier interpretation.
- In a later stage of the project, there should be an "experiment management system" (i.e. a management system which efficiently stores all input data, (pulses, parameter, setup, etc.), results and documentation).
- The system should provide different levels of abstraction: It should be possible to operate directly with pulses (to optimze them or to inspect the physical behavior of the quantum hardware) as well as (later on) representing and executing quantum circuits and algorithms whithout having to mess with corresponding pulse implementation.
- The system should support the execution on real qubit hardware as well as some means of simulation (Integration of the qtip project might be a viable solution for simulation).
- In a later stage of the project, a snapshot feature for runs may be required to allow interrupting and resuming or replaying an experimental run.

Technical / Non-Functional

- The implementation should be object oriented and easy to maintain.
- The migration to other setups or devices should be easy and the interface for both should be as general as possible.

Implementation Language The current software is mostly implemented using the MATLAB script language with some C(++) code for the basic hardware operations. The Bluhm Group wants to move away from MATLAB as the language to set up and run the experiment and favors Python at the moment.

Python - and other interpreted languages with dynamic typing - are great for prototyping and writing small programs fast, which makes them ideal to quickly write a small script that sets up some pulses and executes an experiment without the need to separately compile the program after each small change.

However, these languages tend to result in software that is hard to maintain when used in more complex systems. Especially the lack of static type checks requires the programmer to introduce more sanity checks for parameters and attributes. A compilation phase that is separated from execution often makes it easier to find inconsistencies and errors in the code (before the program is run). Thus, for the implementation of the core functionality of the software, a statically typed language (such as Java, C++, C#, ...) should be used.

Statically typed languages are often a bit more difficult for inexperienced users (or someone who just wants to set up his experiment). Thus, an interface of required functionality should be implemented to allow the usage from scripting languages while the implementation of these features is done in a statically typed language.

C++/C seem to provide the best option for cross-language interface definition, since nearly every interpreted language offers functionality to access C libraries. Additionally, the device drivers of the hardware used in the experiment already come as C libraries. Using these would be easiest in C++/C (this is only a very minor argument, though).

4 Implementation proposition

During several discussions with members of the Bluhm Group (Patrick Bethke and Pascal Cerfontaine), a first implementation idea came up, how to restructurize the pulsecontrol project.

4.1 Usage of the python project "`QtLab`"

Patrick Behtke

4.2 Layers

The software system will be subdivided into layers of increasing levels of abstraction from bottom to top. These layers are shortly introduced in the following and their interfaces are specified in greater detail afterwards.

Driver Layer The bottom-most layer of the software located directly above the hardware drivers of the hardware manufacturers. It defines abstract interfaces for AWGs (Arbitrary Waveform Generators) and DACs (Data Acqusition Cards) and implements these for specific hardware models. These implementations will be as modular as possible and result in units interchangeable by dynamic linking. This layer offers functionality to upload and playback waveforms on single AWGs and read raw measurment data from single DACs.

Hardware Abstraction Layer Located above the Driver Layer, the HAL abstract from the concrete hardware setup. Its interface only offers functionality that operates on virtual channel which are interally mapped on the concrete hardware channels. This mapping is not exposed to higher levels. Datawise this layer also operates on waveforms and raw data.

Data Abstraction Layer This layer abstracts from raw data and provides means to define pulses on a higher level as trees of subelements. It also offers functionality to filter, transform and identify/map measured data.

TimeStream Layer

Quantum Circuit Layer

4.3 The Driver Layer

4.3.1 AWG

For easier man

4.3.2 DAC

4.4 The Hardware Abstraction Layer

4.5 The Data Abstraction Layer

4.5.1 Pulse configuration

4.5.2 Measurement configuration

4.6 The `TimeStream`-representation

A quantum algorithm can be modeled as quantum cirquit. In this model, each qubit has its own lifeline on which gates will eventually appear. This kind of representation lead us to the following implementation idea:

For each `Qubit`, our `Experiment` will have a `TimeStream`-Object, storing time ordered action similar to the quantum circuit. This `TimeStream` object may now contain a list of `Actions`. These `Actions` could be:

- `Initialisations`: pulse or nonpulse changes to the `Experiment`
- `Executions`: simple executions of `Pulses`

- `ConditionedLoops`: You may pass a function and a `Pulse`, this `Pulse` will be repeated, as long as the function returns `true`.
- and many more.

To archive more flexibility, we could also allow to nest `TimeStream` objects or use them in `Action` objects and an `Action` object may also be a `TimeStream` object.

Another possibility of such an implementation is the ability to take snapshots, un- and redo actions. When we track all actions and find an adequate way to store the state of the `Experiment`, we may allow the user to save their work in mid way and continue their work afterwards.

4.7 Quantum Circuit Layer and Algorithm Layer

5 Terminology

AWG : Arbitrary waveform generator, output device.

Pulse : Internal representation of a tension curve.

PulseTab : Pulse defined bye some points with linear integration in between.

Experiment : The set of Input-Data, Measurements and Documentation for each run.

Setup : Internal representation of the links between the output-port of the AWG(s) and the QPU and QPU and the measurement tools.

Waveform : Actual data, that will be played by the AWG.

DAC : Data aquisition card, measurement device.